Optimization-Based Planning

6

6.1 Introduction

The problem of motion planning for autonomous vehicles is to move a set of vehicles within a road segment. As discussed in chapter 'Introduction to Planning', the road segment may be available from higher-level planning, chiefly employing Dijkstra's algorithm (Cormen et al., 2001) for the entire road network graph. A complete road may be broken down into overlapping segments. The chapter considers a general segment of the road which may be occupied by multiple vehicles. The task is to plan and navigate all the vehicles, such that the vehicles do not collide with each other and are able to navigate the road segment in a near-optimal manner while planning in near-real time.

In this chapter, the problem is solved with the assumption of *communication* between the vehicles. Due to safety concerns, it is viable to assume that after autonomous vehicles are introduced and become common on the road, human-driven vehicles would start declining till the point when they become extinct. Considering the safety of other vehicles, human-driving in certain areas may even be declared illegal over the span of time. The benefits of a fully autonomous vehicle transportation system are further encouraging. This chapter explores the use of a Genetic Algorithm (GA) for solving the problem.

The greatest challenge associated with undertaking the task of planning in the absence of speed lanes is the need to deal with a continuous domain of lateral positions. A vehicle may lie anywhere laterally within the road, unlike the lane-oriented traffic in which the lateral decisions are largely about deciding the lane of travel. In such a case, determining the most efficient trajectory which overcomes an obstacle, overtakes a vehicle or simply moves to a better lateral position is a requirement which needs to be fulfilled in a computationally inexpensive manner. Further, because there are multiple vehicles in the road segment, choosing a noncomputationally intensive but *cooperative coordination* strategy is an important requirement. A vehicle must give some other vehicle space to overtake and facilitate some other vehicle to pass through, considering the relative importance and speeds of the vehicles.

Close overtaking is the best example to illustrate the requirement. Considering efficiency concerns over a diverse traffic landscape, every close overtake must be performed. A close-overtaking trajectory is tightly bound on all sides which makes it difficult for the algorithm. The problem is also called the *narrow corridor problem* wherein determining the trajectory is difficult if it goes through a narrow corridor. Further cooperation of the vehicle being overtaken may result in an overtaking attempt being feasible even if the overtaking is narrowly possible. Therefore, the algorithm must allow a vehicle to cooperate and make way for the overtaking vehicle, and then make the overtaking trajectory to carry out overtaking.

On-Road Intelligent Vehicles. http://dx.doi.org/10.1016/B978-0-12-803729-4.00006-4

GA is an *optimization* algorithm which can find optimal value of an objective function by changing the function parameters. The GA is inspired from the process of natural evolution. It imitates the evolution process of humans wherein a population corresponding to multiple individuals is maintained, which undergoes breeding to generate the offspring-making individuals of the next generation. The fittest individuals survive the evolution whereas the weaker ones are eliminated. Similarly, GA maintains a pool of solutions to the problem, each individual representing the encoded solution to the problem in the form of genes. The fitness of each solution is judged by a fitness function. At every generation, specialized genetic operators are applied between the selected individuals to form the next generation of the population pool. The fittest individuals usually replicate with small mutations or deviations, whereas the weaker ones are eliminated. The fittest of the final surviving individual is adjudged as the optimal solution.

The chapter makes use of GA for the purpose. The GA is asked to optimize a *Bézier curve*. The greatest advantage of GA is that it is *probabilistically optimal*, meaning that the probability of optimality tends to 1 as optimization time tends to infinity. It is also *probabilistically complete*, meaning that the probability of finding a solution, if one exists, tends to 1 as the optimization time tends to infinity. GA fails when the optimal trajectory has too many turns or the trajectory goes through a very narrow region. Every turn is encoded as a gene in the GA and too many turns means optimization with too many variables. The GAs encounter the problem of the *curse of dimensionality*, meaning that the performance drastically reduces on an increase in the dimensionality or the number of variables optimized. Further, it is very difficult to generate samples inside narrow corridors, meaning that it is hard to sample trajectories that pass through a narrow region tightly bounded by obstacles on all sides. The sampling-based approaches generally fail in such scenarios.

Road scenarios are marked with fewer steering manoeuvres and too-narrow regions defy safety concerns and are hence undesirable. Hence, for this problem GA is a good choice of algorithm. Further, the iterative nature of GA makes it very desirable for such problems. It is an *anytime algorithm* and a near-optimal solution to the problem is available at any time in the optimization process. The solution improves with time. The near-optimal solution can be extracted at any moment for the immediate motion of the vehicle, and hence the algorithm is best suited for such problems in which a small and usually fixed optimization time is available.

The biggest disadvantage of GA is, however, its *computational cost*. Sections 6.4 and 6.5 highlight the manner in which the problem characteristics can be exploited to make it possible to run GA on such a real-time problem. The other problem is the presence of multiple vehicles on the road which requires an optimal *coordination strategy* working in near-real time. Because the general coordination strategies like priority-based planning (Bennewitz et al., 2001, 2002), co-evolutionary GAs (Potter, 1997; Stanley and Miikkulainen, 2004) or any other related techniques may either not be optimal or be computationally intensive, the requirements are difficult to keep. The work uses *prioritized-based coordination* which is noncooperative but computationally inexpensive. Every vehicle is assigned a priority and the vehicles are planned strictly in the order of priority. A lower-priority vehicle is responsible for collision

avoidance with a higher-priority vehicle. *Traffic-inspired heuristics* are embedded into the strategy to make it near optimal and cooperative, at the same time being computationally inexpensive. To do so, the general operation of the traffic is observed, prevalent traffic and social rules are assessed and methods are designed by which these rules can be coded over the GA to make a coordination strategy.

Segments of different sections including text and figures have been reprinted from Kala and Warwick (2014), Applied Soft Computing, Vol 19, R. Kala, K. Warwick, Heuristic based evolution for the coordination of autonomous vehicles in the absence of speed lanes, pp. 387−402, Copyright (2014), with permission from Elsevier.

6.2 A Brief Overview of Literature

Most works in the domain of motion planning for autonomous vehicles exist for organized or a lane-oriented traffic. Some notable works, however, also exist for unorganized traffic (or extendable for unorganized traffic), which is the key focus of the chapter. Chu et al. (2012) constructed a number of candidate paths from which the best path was selected. The strategy can be used for avoiding obstacles by a single manoeuvre only. Paruchuri et al. (2002) simulated the vehicle behaviours on straight roads and crossings without traffic lights. The limitations include no cooperation between the vehicles and that the overtaking decision module does not generalize to a high number of vehicles with unorganized patterns.

The algorithms used in robot motion planning may be of use for planning autonomous vehicles in the absence of lanes, although they may require to be remodelled per the traffic scenario. Optimization-based methods are widely used for planning the trajectory of a single robot or multiple robots. Many of these approaches are, however, offline and cannot be executed in real time. The use of the road coordinate axis system instead of the Cartesian coordinate axis system can be very helpful when using these algorithms for autonomous vehicles. Many of these approaches are for a single robot only and hence noncooperative. Other approaches with multirobot coordination techniques are largely computationally intensive.

A number of good approaches using GA can be found which are all offline and hence cannot be directly used. Xiao et al. (1997) planned the path of a robot offline by using a GA. The path was also updated by an online planner. Xidias and Azariadis (2011) solved the problem of integrated routing and planning of a group of vehicles. A centralized approach was proposed and all travel information parameters were embedded in a single chromosome. Kala et al. (2011) used a multiresolution collision-checking approach to compute the trajectory of a single robot using a GA. The resolution was coarser at the start of the GA optimization, which became finer as the GA optimized the path. These approaches are largely offline and noncooperative.

Garcia et al. (2009) used the Ant Colony Optimization Algorithm along with a memory unit to restrict the magnitude of exploration in search for a path. The algorithm worked for discrete spaces only. In another related work, Lepetic et al. (2003) designed planning algorithms for robot soccer. Scenarios were precomputed in terms

of relative positions of ball with respect to the robot. The intermediate solutions were generated using interpolation. In the problem of autonomous vehicle navigation, precomputation is not possible due to the discrete decisions between overtaking and not overtaking.

Some good works also exist for multiple robots using centralized optimization, which can be computationally expensive for a large number of robots. Lian and Murray (2003) solved the problem of motion planning for a single robot, a swarm of robots and uncoordinated multiple robots using quadratic programming and spline curves. Similarly, Klancar and Skrjanc (2010) used Bernstein−Bézier curves which were optimized to produce short path length and high clearance. Kapanoglu et al. (2012) used optimization for the problem of area coverage using rectilinear moves. Szlapczynski and Szlapczynska (2012) also used optimization for planning the trajectory of a ship.

Some variants use cooperative co-evolution for time efficiency, which is still too computationally intensive to be applied for real-time applications. Kala (2012) used cooperative co-evolution for motion planning of multiple robots moving rectilinearly in narrow corridors. A memory unit was proposed for sharing the shortest-path information between landmarks. Wang and Wu (2005) also made use of cooperative co-evolution. Chakraborty et al. (2008) solved the problem using both centralized and decentralized methodologies using differential evolution. Only the next immediate step of the robots was optimized. The approach was noncooperative.

6.3 A Primer on Genetic Algorithm (GA)

The first and the simplest tool used to solve the problem is the GA. The GA is an *optimization*-based technique. That said, the GA can do a lot more than conventional optimization; however, to ease the discussion GA is studied as an optimization tool in this section and then modelled differently to solve the problem of motion planning for autonomous vehicles in Section 6.4. Optimization techniques are used to find the minimum or the maximum value of an optimization *objective function*. The techniques do so by changing the parameters which affect the value of the optimization objective. The general nature of the optimization problems is given by Eq. [6.1].

$$\text{Calculate Min}(F(x_1, x_2, x_3 \ldots x_n)) \qquad [6.1]$$

$$\text{LB}_i \leq x_i \leq \text{UB}_i$$

Here, $F()$ is the optimization objective function with a total of n parameters. Each parameter x_i is bounded between a lower bound (LB_i) and an upper bound (UB_i). There may be more linear or nonlinear constraints.

The GA is inspired by the principle of *natural evolution*. In natural evolution, numerous individuals of every species exist in the ecosystem. All the individuals collectively are known as the population pool. The individuals of the population pool interact with each other during their life cycle. The interactions may be more

with the individuals closer or alike, and less with the others. The individuals are essentially chromosomes which are a collection of genes. The genes together make up all the properties of the individual and completely characterize the individual. The individuals mate and produce new offsprings who constitute the future individuals of the population pool. Typically, two parents mate to generate the next generation of children. The children contain the mixed characteristics or genes from the parents, with some genes derived from the first parent and other genes derived from the second parent. While doing so, however, many times genetic errors may happen resulting in different or new genes in the children, produced as a result or error while copying the specific genes. These errors are very small as compared to the total collection of genes. Out of all the children generated, the fittest survive whereas the weaker cannot survive in the hostile environment and die in the process. This is in accordance with Darwin's theory of evolution. The surviving children constitute the next generation of the population pool. In this manner evolution happens generation by generation, and the latter generations are fitter and more suited to the changing environment.

6.3.1 General Algorithm Framework

As an imitation to the natural evolution process, the GA attempts to solve the problem by generating random solutions to the problem or by random assignment to the parameter values. Each such solution is called an *individual*. The solution in the problem-specific domain is called the phenotype. The phenotype needs to be encoded in the form of genes making the chromosome. This encoded solution is called the genotype. The *population* is a collection of individuals at any particular point of time or *generation*. The GA is hence a multiindividual method in which the search for the optimal solution happens by employing multiple individuals which work in coordination with each other. The initial population may be randomly generated. It is also possible to use heuristics to analyse the problem domain to identify some good characteristics or partial solutions which are likely to have a good fitness value. Correspondingly, the initial population may be generated biased towards some values of genes.

The individuals at any generation are subjected to a number of *genetic operators* to create the individuals of the next generation of the population pool. Essentially, the fittest individuals are selected and *crossover* is applied to get the children. The fittest individuals are likely to generate a better solution and are selected more often than the weaker solutions, which may not be selected at all in the evolution process. The crossover operation generates children with gene values intermediate between the parents. Some children are additionally subjected to *mutation* wherein small changes are applied to their gene values. This makes the population pool of the next generation.

Here, *fitness* is an assessment of the goodness of the solution for the particular problem. The fitter be an individual, the more is its chance to replicate and dominate its characteristics in the population pool. The genetic operators try to generate individuals with a better fitness value. The *fitness function* hints the evolutionary process towards which characteristics or genes to prefer, and which ones not to prefer. The overall process is shown in Fig. 6.1.

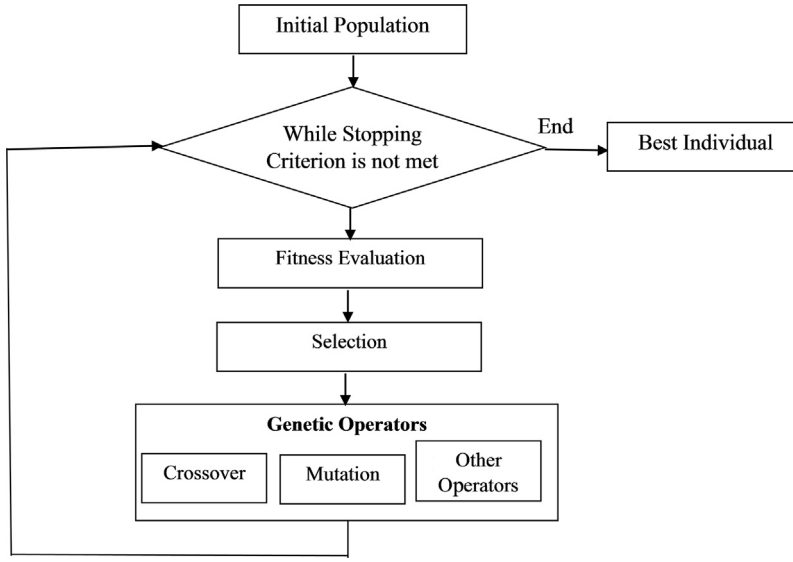

Figure 6.1 Genetic algorithm (GA).

6.3.2 Individual Representation

The purpose of *individual representation* is to encode the phenotype representing the real solution to the problem represented as a solution in the actual problem domain, into a genotype or the representation which the GA can handle. This representation is in the form of a collection of genes. The representation technique should facilitate the conversion of the genotype into phenotype for fitness assessment and for porting the final optimized solution into the problem domain. Similarly, conversion of phenotype into genotype may be needed to insert specific individuals in the initial population pool.

Each gene constituting the individual may be binary, only storing 0 and 1. The GA resulting from such a representation is called a *Binary* GA. For the above generic definition of the problem, each variable x_i representing a real value must be converted into a binary equivalent for the GA to work. The number of bits used for every variable must be fixed which constitutes the resolution of the algorithm. On the contrary, every gene may also be allowed to store real numbers. The GA resulting from this representation is called a *Real Coded* GA. In this work the stress is on Real Coded GAs, which are easier to understand and develop. For the generic definition of an optimization problem, the individual is simply all the variables appended one after the other shown in Fig. 6.2.

0.54	0.04	0.87	0.65	……	0.39
x_1	x_2	x_3	x_4	…….	x_n

Figure 6.2 Individual representation.

A tree-based representation is another popular choice for representing individuals, wherein the genotype is in the form of a tree with each node presenting a parameter and its relation. Genetic programming is a dominant technique which uses such a tree-style representation, designed for automatic evolution of programmes which inherently have a tree-like structure. Specialized operators are made to work with tree-based representations. It is also common to give a linear encoding to a tree-based representation to suit conventional genetic operators. Grammatical evolution is a common technique. Further, many times the algorithm metaparameters may also be embedded along with the parameters representing the variables of the optimization objective. This gives a self-adaptive nature to the approach wherein the metaparameters are also optimized. Evolutionary strategies is a common technique using this principle.

For this problem the representation and conversion is simple. For the problem of trajectory planning, the phenotype represents a trajectory usable by the vehicle whereas the genotype is a small set of real numbers. This problem is addressed in Section 6.4.

6.3.3 Genetic Operators

The purpose of genetic operators is to generate the next-generation population from the current generation population. The different operators are described in the following subsections.

6.3.3.1 Scaling

First, every individual in the pool must be assigned an expectation value which denotes its likelihood of being selected in the evolutionary process. This operation is known as *scaling*. The expectation value can be easily taken proportional to the fitness value. A normalization may be required to get the range of values within the desirable range. This is called *fitness-based scaling* wherein the expected value ($E(I)$) of any individual (I) is given by Eq. [6.2].

$$E(I) = E_{\min} + \frac{F(I) - F_{\min}}{F_{\max} - F_{\min}} (E_{\max} - E_{\min}) \quad [6.2]$$

Here, $E_{\max}$ and $E_{\min}$ are the maximum and minimum desired expected values, whereas $F_{\max}$ and $F_{\min}$ are the maximum and minimum fitness value returned by the fitness function.

The problem with fitness-based scaling is that very soon a very few individuals get very good fitness and correspondingly a very high expected value. Hence, they get selected again and again, whereas the others get eliminated not being selected for the evolution. This causes a *premature convergence* in the population pool, wherein very soon all individuals have nearly the same characteristics. Hence, an alternative method is adopted to stop a few individuals from dominating the population pool very early. Here, the expectation value is kept proportional to the rank of the individual and the technique is known as *rank-based scaling*. Even if a few individuals have significantly better fitness as compared to the rest of the individuals in the population

pool, the expectation is assigned based on the ranks alone, the distribution for which does not change. The expectation value is given by Eq. [6.3].

$$E(I) = E_{\min} + \frac{N - Rank(I) - 1}{N - 1}(E_{\max} - E_{\min}) \quad [6.3]$$

Here, $Rank(I)$ is the rank of the individual I in the population pool with rank 1 denoting the best individual. N is the population size.

The *top scaling scheme* selects the top few individuals and gives them equal expectation values. The top few individuals hence have equal chances of being selected, whereas the others get eliminated.

6.3.3.2 Selection

The *selection* operator actually selects the individual for the evolution process. The individuals with high fitness value may be selected multiple times, whereas the weaker ones may not be selected at all. The individuals not selected are simply deleted. The selection is a stochastic process with the possibility of selection given by the expectation values. The implementation of this concept is different for different selection schemes, a few of which are briefly discussed.

The *roulette wheel selection* scheme makes a roulette wheel with different individuals as different sectors of the wheel. The selection is summarized by Fig. 6.3A. The possibility of selection of a particular sector in a roulette wheel is given by its circumference, which is kept as the expectation value of the individual. The wheel is rotated once and the individual to which the pointer points when the wheel becomes stationary is selected. For selection of multiple individuals, the wheel is rotated multiple times and each rotation gives a selected individual.

The problem with this scheme is that the fitter individuals occupy a large circumference and are likely to be selected multiple times leading to a premature convergence. Hence, a different selection technique known as *Stochastic Universal Sampling* is used. Here, the roulette wheel has as many pointers or selectors as are the number of individuals to be selected from the population pool shown in Fig. 6.3B. Each pointer selects one individual. For selection of multiple individuals, the wheel is rotated only once and the individuals corresponding to all the pointers are selected. Of course, the fittest individuals may be selected multiple times with multiple pointers pointing to that particular individual.

Another common selection scheme is *tournament selection*. Here, tournaments are organized between a few individuals. In any tournament between two individuals, the probability of winning of the fitter individual is kept as p, whereas the probability of winning of the weaker individual is kept as $1 - p$. The winner of the entire tournament is selected.

6.3.3.3 Crossover

The *crossover* operator imitates the reproduction of the natural species by making children from their parents. The children always carry the characteristics of their parents

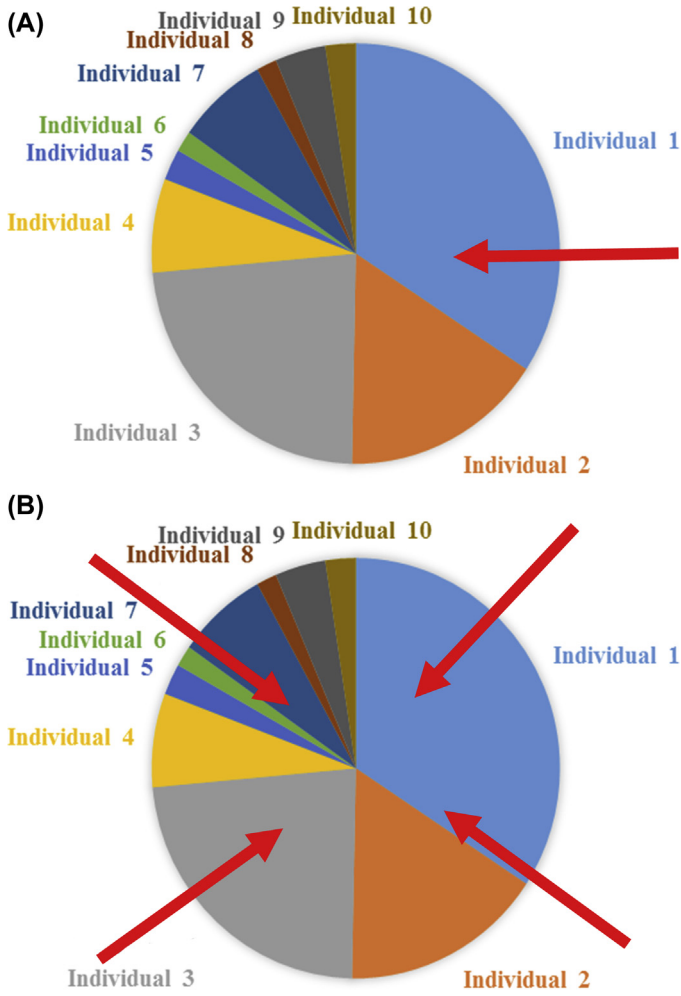

Figure 6.3 Selection (A) roulette wheel selection, (B) stochastic universal sampling.

and are formed by the exchange of genes from the parents. Any number of parents may be used to produce the children, although the most common example is a *binary crossover* in which two parents are used to produce two children. The multiparent crossover operators extend the same idea by exchanging genes between multiple parents to produce children. The working of binary crossover is discussed here. The higher-order crossovers are a natural extension to the same principles. The ratio of the number of individuals selected for crossover to the total population size is called the *crossover rate*. In other words, crossover rate denotes the proportion of individuals which undergo crossover.

The simplest crossover is the *one-point crossover* wherein a random point is selected in the chromosome of the parents. The parents are placed on top of each other

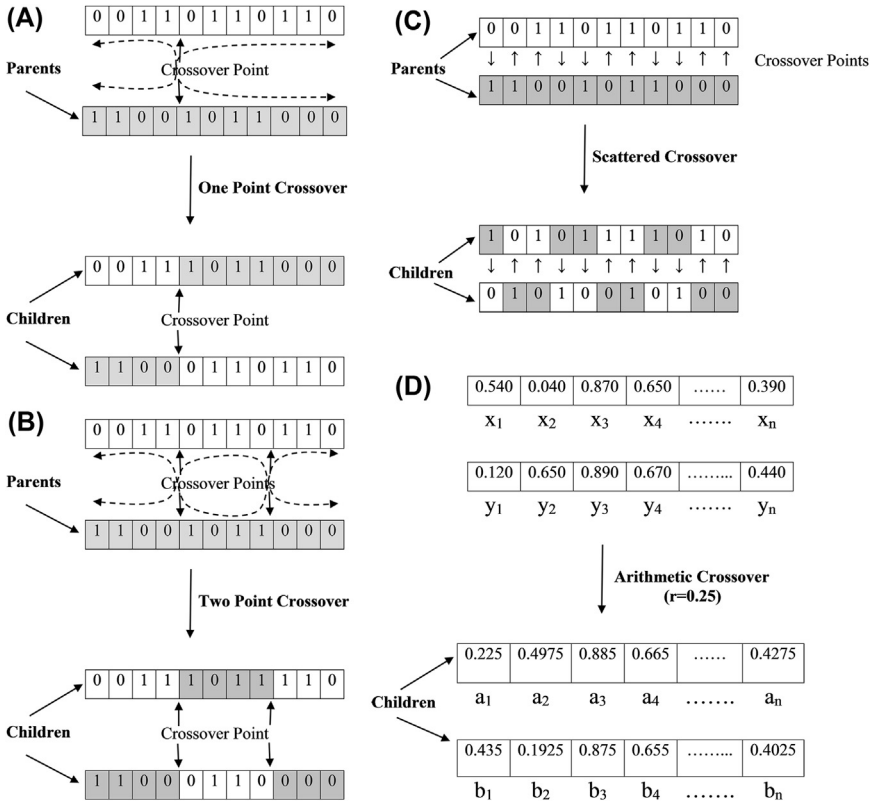

Figure 6.4 Crossover (A) one-point crossover, (B) two-point crossover, (C) scattered crossover, (D) arithmetic crossover.

and the genes lying after the crossover point are exchanged. This gives rise to two children from two parents. The technique is shown in Fig. 6.4A. Similarly, *two-point crossover* selects two random crossover points instead of one. The parents are kept on top of each other and the genes in-between the two crossover points are swapped. The technique is shown in Fig. 6.4B. Both techniques have the problem that genes located close to each other are likely to go together to the same child. Hence, a *scattered crossover* technique is used. Here, half the genes are randomly copied from the first parent to the first child, whereas the other half is copied from the second parent to the first child. The second child takes the other genes. It behaves like multipoint crossover in which the number of points are equal to the chromosome length and at each point a random decision is make whether to swap the genes or not. The technique is shown in Fig. 6.4C.

Another crossover technique used largely for the real coded GAs is the *arithmetic crossover*. Each individual represents a point in a multidimensional search space (see Section 6.3.5). The two parents are two such points in the search space. This crossover technique attempts to generate children in-between the line joining the two parents.

Using this technique the *i*th gene of the children *A* and *B* with parents *X* and *Y* is given by Eq. [6.4].

$$a_i = rx_i + (1 - r)y_i \qquad [6.4]$$

$$b_i = (1 - r)x_i + ry_i$$

Here, *r* is a random number between 0 and 1. The technique is shown by Fig. 6.4D.

6.3.3.4 Mutation

The *mutation* operator tries to bring into the population new characteristics or gene values which are nonexistent in the population pool. The addition of new characteristics to the population pool may be good, in which case the mutated individual enjoys a good fitness value and is bound to be selected multiple times, or the addition of new characteristics may be bad leading to the individual being eliminated from the population pool. This makes the individuals explore in search of the optima by constantly changing the gene values.

The *mutation rate* decides the magnitude of changes to be made in an individual to produce the mutated individual which constitutes the individual of the next generation. In a binary GA, the gene is simply flipped with 0 changed to 1 and 1 changed to 0, with a probability given by the mutation rate. The mutation is given in Fig. 6.5A. In real coded GA, the maximum percent change made in the value of any gene is given by the mutation rate, whereas the actual deviation is produced by a random number. It is advisable to keep the mutation value small so as not to cause randomness to the search process, whereas too small mutation rates may hardly make any changes to the individual resulting in very slow convergence. A popular technique is thus to use *Gaussian mutation* in which the magnitude of deviation to make in an individual is given by a Gaussian distribution which has the inherent property that the small

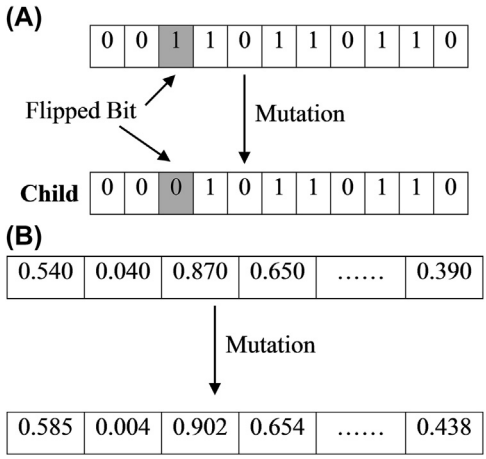

Figure 6.5 Mutation (A) bit mutation, (B) gaussian mutation.

values are most likely and are often generated, at the same time the higher values are also rarely generated. The mutation applied to a synthetic individual is shown in Fig. 6.5B. Alternatively, it is common to make two mutation operators called *soft mutation* with a small mutation rate which is called very frequently and *hard mutation* with a very large mutation rate which is called rarely.

6.3.3.5 Other Operators

Depending upon the problem and its characteristics, a number of other standard or nonstandard genetic operators may be designed. A common operator is *elite* which identifies the top few individuals of every generation and passes them straight to the next generation. It is sometimes possible that the best individual may not survive in the selection, or even if it survives, may be mutated to reduce its fitness. The elite operator ensures that the best individual is safe and hence the fitness of the best individual does not deteriorate for a constant fitness function.

Similarly, if the problem has constraints it may be possible to get individuals which do not adhere to the constraints and hence cannot be accommodated in the population pool. A common technique is to introduce a *penalty* in the fitness value of these individuals so that they are no longer selected. This also creates evolutionary pressures to reduce infeasibility and ultimately eliminate it. Alternatively, one may make a custom *repair* operator which is applied to all infeasible individuals. This operator changes an infeasible solution to a feasible one by changing its genes such that the feasibility condition is met. In the specific problem of trajectory planning, a trajectory hitting an obstacle is a constraint violation which can be handled in this manner.

Another operator commonly used is *insert* which inserts new individuals to the population pool. A common use of this operator is when the fitness function changes with time and can show trends of sudden change. In such a case, random solutions introduce an intent to create enough *diversity* in the population pool so that there are always individuals with characteristics to react to and survive any change of fitness function. The random solutions are also useful to add new characteristics to the population pool to eliminate premature convergence.

6.3.4 Stopping Criterion

The GA may go on indefinitely improving the individuals along with the generations. However, the algorithm needs to be terminated after some time so that the optimal solution can be extracted. At the later generations, *convergence* happens when all individuals have nearly the same characteristics, and it is extremely unlikely to create new characteristics that aid the optimization. It is hence better to terminate the algorithm when convergence happens.

The *stopping criterion* states the condition on meeting which the optimization terminates. Multiple criteria may be set in which case any of them must meet to terminate the optimization process. Common criteria include limiting the maximum number of generations, maximum optimization time, stall generations or number of generations

in which no improvement is seen in the GA and stall time in which no improvement is seen in the GA.

6.3.5 Exploration and Exploitation

Consider the objective optimization of Eq. [6.1] which can easily be taken as the fitness function of the GA. If this fitness function is plotted, it would be a surface in a high dimensional $(n + 1)$ space with n axis representing each of the variables and 1 axis representing the output. This is known as the *fitness landscape*. Because a multidimensional curve cannot be drawn in the book, let us consider a function with two variables which can be easily drawn as a 3D surface and as a contour. Let these be given by Fig. 6.6. Each individual is a point in this contour with the fitness value given by the value of the contour. The population pool is a group of points in this fitness landscape. Initially, the individuals are randomly generated and so the points are randomly distributed in the fitness landscape. As generations pass, the individuals in the high-fitness areas replicate by the selection operator and attract other individuals towards them by the crossover operator. The individuals in the poor-fitness areas are eliminated by the selection process. The mutation operator causes individuals, especially those in the high-fitness areas, to explore their surroundings by taking a small step at a random direction.

Exploitation refers to the evolutionary forces or the genetic operators which cause the individuals to come close to each other and converge at some point in the fitness landscape which is usually the global optimum. *Exploration* refers to the evolutionary forces or the genetic operators which cause the individuals to go away from each other

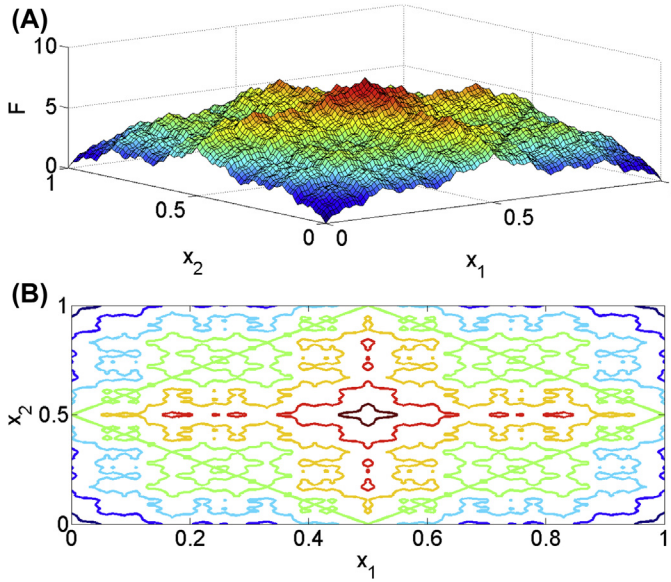

Figure 6.6 Fitness landscape. (A) Complete Space, (B) Contours.

in pursuit of, or search for, the global optimum. Normally, exploitative pressures are kept larger than the explorative pressures which cause the population pool to contract and eventually converge to a very small region or the global optimum.

Attempting to cause very early convergence by parameter settings can lead to premature convergence into a local optimum as much explorative efforts were not made to search for the global optimum. Very small exploitation can lead to no convergence or a very small convergence which necessitates the need to run the algorithm for a prolonged time. Absence of convergence makes the algorithm behave similar to a random search algorithm, which searches by continuous generation of random individuals. It is important to keep a judicious balance between the two forces, whereas the computational time is a major decider between the contributions of the two forces. Availability of long computational time means that time can be spent on explorative settings, whereas the need to give results in very short computational times will necessitate exploitative settings.

The problems are magnified when the fitness landscape has high *modality* or too many crests and troughs. Too many local optima can deceive the evolution resulting in convergence at the local minima or generating evolutionary pressures near the local minima. Large modality calls for large exploration to sight the global optimum. If possible, choosing fitness functions with limited modality is an important criterion in the design of fitness functions.

Based on these discussions, it is intriguing to study the effect of different parameters of the GA. The larger the mutation rate, the greater is the exploration, demanding more computational time at the benefit of missing a local optimum. Similarly, large crossover rates may result in exploitation and premature convergence. Keeping the same number of individuals in the population, or the population size, results in better optimization due to more exploration at the expense of computation time. If the number of generations is, however, reduced to keep the time constant, the algorithm will become more explorative. Similarly, carrying optimization for a large number of generations results in better optimization with more computational expense. Cutting down on the population size to keep the computational time constant can be an exploitative setting.

6.4 Motion Planning with Genetic Algorithm

This section makes use of GA for the problem. Before selecting any algorithm for the problem, it is always wise to consider its pros and cons. The pros and cons of GA are summarized in Box 6.1. The challenge in the work is to plan the motion of the vehicles without the assumptions of lane-based driving. Not considering lanes necessarily increases the problem of planning and coordination to a high level but provides for a more general solution. The most important problem faced here is in overtaking, due to the requirement for safety under optimal steering conditions, resulting in a small margin for error in the efficiency of driving. However, the vehicle may ask other vehicles for support to make a close overtake possible. Such a solution is inspired from

Box 6.1 Pros and Cons of Genetic Algorithms (GAs)

Pros
- Probabilistically optimal
- Probabilistically complete
- Iterative

Cons
- Computational cost
- Do not work well in narrow corridors
- Do not work well with too many turns (implying large dimensionality)
- How to introduce cooperative coordination?

observed driving in low-width high-density roads, especially where vehicles vary in speeds to a large degree.

Whilst driving along straight roads may be a relatively straightforward affair, making efficient turns for most crossings or other natural turnings of the road, requires expertise. Turning too close to the inner boundary may require reducing the speed, at the same time turning along the outer boundary may make the path too long and suboptimal. Here, the same is done by using *Bézier-based motion planning* linked with a GA for the optimization procedure. The GA is adapted to work on real-time scenarios by using a space−time-approach model of different vehicles and a globally referenced individual representation. A GA is probabilistically optimal and probabilistically complete and can work in continuous spaces. The objectives of efficiency and safety can be easily knit to a single objective function for evolutionary optimization. Road scenarios have limited possible homotopies for a vehicle, which translates to a limited modality of the fitness landscape (given the fact that a *repair operator* is designed which maps every possible trajectory to a reasonable looking, feasible and short trajectory). For a limited-modality fitness landscape, the optimal solution can be found reasonably early or, to put it another way, the probability of finding the optimal solution is high. Here, assumptions are made on a limited number of obstacles and other vehicles, and with a limited-resolution map.

The algorithm is a real-life application of *dynamic evolutionary algorithms*, which optimize an objective under a changing fitness landscape. The operational scenario continuously changes as the vehicles move. At every instant the latest instance of scenario is given to the GA for optimization. A single population pool of the GA is maintained for a vehicle, which adapts to the changing scenario. This is an example of *incremental learning* in which the learnt model (GA population) has to be incrementally updated against the changing data (scenario). The changes may be gradual requiring the GA to incrementally learn by small changes to the model (GA population), or the changes may be large effectively requiring reworking of the complete model. Enabling optimization to be carried out in real time involves a variety of challenges, which are handled by various methods in this chapter. As the solution is

designed for a specific application, deterministically adaptive strategies can be framed for each of the challenges.

Traffic rules of everyday driving can in fact play a major role in *coordinating* the motion of multiple vehicles in general scenarios. Rules, such as driving on the left- (or right-hand side of the road, and overtaking on the right (or left)), play a major role in enabling drivers to plan their motion amidst multiple vehicles. Although tackling the complete problem of vehicle coordination would be extremely large, it is observed that embedding these rules as heuristics can play a major role in realizing an overall efficient strategy. No off-the-shelf solution to coordination is available from the mobile robotics literature which can guarantee optimality, cooperation and short computational times, as those solutions do not use these traffic heuristics.

The problem considered here consists of motion planning of N robotic vehicles. Each robotic vehicle R_i has a predefined source S_i and a predefined goal G_i. It is further assumed that the specification of all the roads, in terms of their start, end and connectivity with other roads in the geographical map, is known a priori. The planning must ensure that each vehicle reaches its destination, and that there is no collision with any other vehicle along the way. The road may have static obstacles which can all be sensed as and when they appear.

For simplicity, it is assumed that all vehicles are rectangular in shape with length l_i and width w_i. Every vehicle starts from its source S_i at a time T_i and reaches its goal G_i. The vehicle is assumed to disappear after its journey has been completed. At any time t during its journey, the vehicle is at a position X_i (x_i, y_i) measured in terms of global coordinates, moving with a speed of v_i in a direction of θ_i.

The approach developed here for solving the multivehicle coordination problem (see Fig. 6.7) consists of Dijkstra's algorithm for path selection; this enables the vehicle to reach its goal from the source. The algorithm works over the road map, which is known for any city with all the roads and intersections well identified. This step is common to all algorithms presented in this book and has already been explained in chapter 'Introduction to Planning'. The algorithm can also detect road blockages which lead the vehicle to replan, again using Dijkstra's algorithm. As a result, the coarser path always points to the valid path at the coarser level being tracked by the vehicle. The next level of planning consists of Bézier -curve planning, which is optimized by a GA (Section 6.4.1). The GA uses a set of genetic operators (Section 6.4.2) to compute the optimal path measured by the fitness function (Section 6.4.3). This level deals with uncertainties associated with the relative positions and speeds of the other vehicles and road boundaries. The output of the planner at any time step is the Bézier curve or a trajectory, which is used to move the vehicle. The source of the finer-level planning is always the current vehicle position, whereas the goal is the next distant crossing in the coarser path. As the vehicle approaches its current goal, so the goal is updated to the next crossing. In this manner, the coarser path constantly guides the finer-level planning by making it reach one crossing after another up to the point at which the vehicle's final destination is reached.

Planning is performed independently for all vehicles. Coordination dictates the manner in which the vehicles avoid each other, whereas each vehicle is planned using finer-level planning. The developed coordination strategy (Section 6.5) states that a

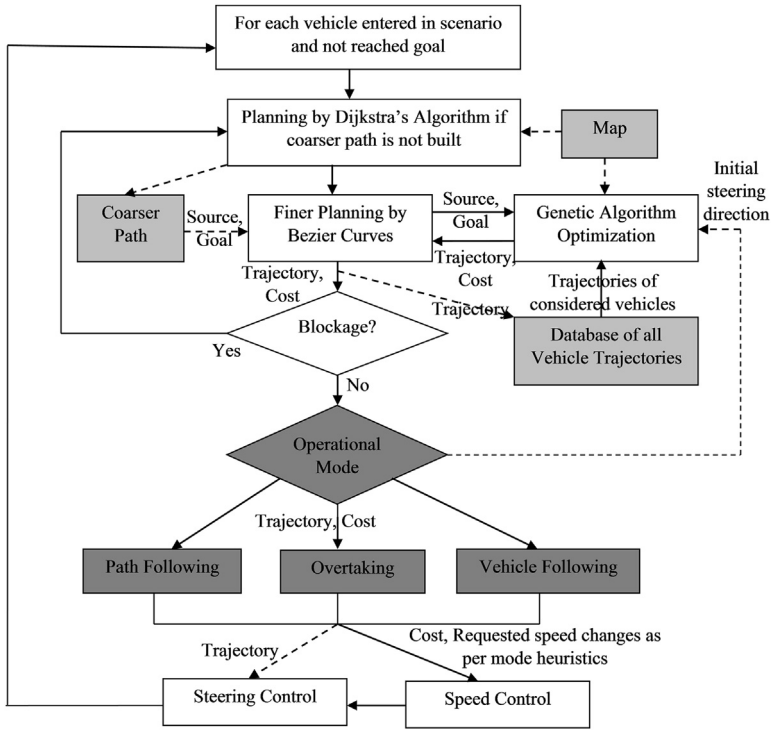

Figure 6.7 General planning algorithm using genetic algorithm (GA).

vehicle only considers the vehicles ahead in its planning which is in consensus with natural human driving wherein one largely considers the prospective motions of the vehicles in front to make one's navigation plan. The vehicle trajectories are represented as a space−time graph. While planning, a vehicle to the rear treats all vehicles ahead as dynamic obstacles the dynamics of which are known. Every vehicle attempts to move by its highest possible speed (Section 6.5.1).

The resultant coordination strategy cannot display cooperative overtaking, which is handled by plugging in an additional measure in the coordination strategy, according to which a vehicle may ask another vehicle to make some initial turn or alter its speed to better suite its plan. The initial turn is handled by the GA, whereas speed is handled by the speed control module. Two heuristic strategies are designed that make use of this measure, namely overtaking (Section 6.5.2) and vehicle following (Section 6.5.3). Both these strategies assess the current scenario and heuristically compute a speed and steering assignment for the different vehicles. The general traversal in the absence of a vehicle ahead is by the path-following strategy.

The key features of the approach are summarized in Box 6.2. The overall approach is also summered in Box 6.3.

First, Dijkstra's algorithm is used to compute the route of the vehicle. For any vehicle R_i, the output is a set of crossings (or vertices) that it must traverse in strict order. Hence, the path of R_i becomes $S_i \rightarrow P_i^1 \rightarrow P_i^2 \rightarrow P_i^3 \rightarrow \ldots \rightarrow P_i^v \rightarrow G_i$. The optimality

Box 6.2 Takeaways of Solution Using Genetic Algorithms

- The design of a GA which gives results within *short computational times* for traffic scenarios.
- Employment of the developed GA for *constant path adaptation* to overcome actuation uncertainties. The GA assesses the current scenario and takes the best measures for rapid trajectory generation.
- The use of *traffic rules* as heuristics to coordinate between vehicles.
- The use of heuristics for constant adaptation of the plan to *favour overtaking*, once initiated, but to cancel it whenever infeasible.
- The approach is tested for a number of diverse behaviours including obstacle avoidance, blockage, overtaking and vehicle following.

Box 6.3 Key Aspects of Solution Using Genetic Algorithms

- Use *road coordinate axis system* for better sampling
- Optimize *as the vehicle moves*
  - Tune plan
  - Overcome uncertainties
  - Compute feasibility of overtake
- Individual representation
  - Variable number of turning points with lateral and longitudinal component
  - Source, first directional maintenance point and goal are fixed
  - Points always *sorted* as per lateral axis
  - All points *longitudinally ahead* of the current position of the vehicle
  - Trajectory is the *Bézier curve* from these points
- Genetic operators
  - *Repair*, sort and delete points behind current position, delete excess points to shorten/smoothen the path.
  - *Insert* random individuals
  - *Crossover*, suited for variable-length chromosomes
  - *Mutation*, randomly deviate points
- Fitness function
  - Trajectory length
  - Length of infeasible trajectory
  - Length of trajectory without safety distance

of Dijkstra's algorithm is based on the assumption that different roads have similar traffic. The road network consists of a large number of vehicles in all, and jointly analysing all of them for the overall optimal plan is not possible in real time due to the amount of necessary computation.

As the environment is dynamic, the computed path may, at any instance of time, be found to be *blocked*. The algorithm must, in this case, take alternative means to act in

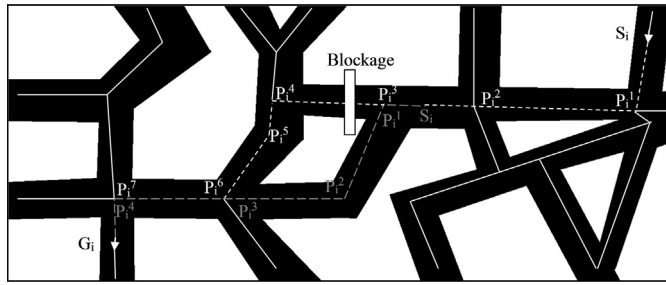

Figure 6.8 Road map with blockage. Each point of intersection of two lines (of paths) is a vertex. The shortest path from source to goal is shown as a dotted line. The list of vertices is returned by Dijkstra's algorithm (S_i, P_i^1 to P_i^7, G_i) shown in white. The path may change upon detection of a blockage. The new path is shown in grey.

response to such blockages. This process consists of blockage detection and replanning. In this algorithm, it is assumed that if the next level of planning algorithm failed to generate feasible solutions for consecutive *block* iterations, then the path is regarded blocked. In such a scenario the current position of the vehicle is added as the new source S_i as a new vertex to the road map, the goal is left unchanged. The corresponding connections and weights are updated, to account for the blockage. Dijkstra's algorithm is called again to return a new path. The movements now take place as per this path, as is shown in Fig. 6.8. In the case that no feasible path is possible, the journey of the vehicle is terminated.

The next (finer) level of the planner is carried out by the use of *Bézier curves* (Bartels et al., 1998). The purpose of the planning algorithm is to generate a path which the vehicle may follow using its own control mechanism. The generated path needs to be as smooth as possible, so that the vehicle may be controlled maintaining high speeds as per the nonholonomic constraints. The other purpose in the use of Bézier curves is to enable the vehicle to make efficient turns in all scenarios of crossing, general road turn or while overtaking. This is shown in Fig. 6.9.

The GA planning algorithm needs its own source and goal for planning. The source is always the current position of the vehicle (X_i with the vehicle oriented at an angle θ_i). At every time step, the planner generates a trajectory and the vehicle moves as per the same. At the next time step, the changed position becomes the source for the planning. The goal is fixed as the next crossing P_l^j, that is at least η units apart from the current position of the vehicle X_i. As the vehicle continues its journey using this plan, as soon as it comes close enough to the directed goal P_l^j, the goal is changed to the next position in the vehicle's path P_l^{j+1}. However, this change is not performed in the case when P_l^j is the final goal (Kala et al., 2010a). P_i^k always denotes the crossing just behind the vehicle.

6.4.1 Individual Representation

A GA is used to generate an optimal Bézier curve for motion. One of the major tasks in the use of the GA is to devise an *individual representation strategy*. The Bézier curves

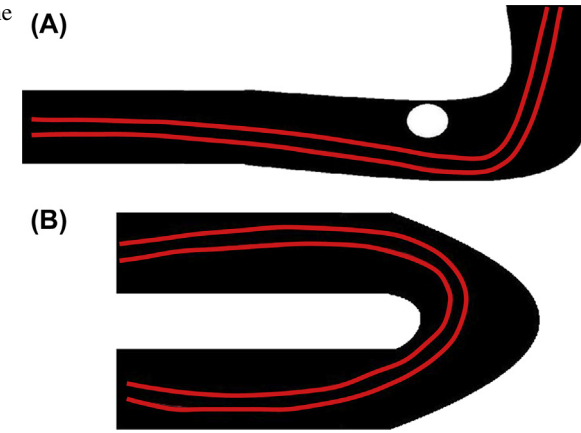

Figure 6.9 Path generated by the vehicle in multiple scenarios using GA. A vehicle at every instant in time generates a feasible path as per the given map, considering the other moving vehicles (not shown in the figure). (A) Scenario 1, (B) Scenario 2.

may be easily generated by means of *control points* in the map. For a better solution in the generation of feasible paths, the road axis system is taken.

The complete Bézier curve path specification (or genetic phenotype) consists of (in order) the source X_i, direction maintenance points, a variable number of additional Bézier control points (subjected to a maximum of P^{max}) and the goal. The directional maintenance points are added to assure that the Bézier curve generated is initially inclined in the direction the vehicle is currently facing or θ_i. Out of all these points constituting the Bézier curve, only the Bézier control points are nonfixed which are hence the points optimized by the GA. All the control points are placed one after the other to form the genotype. Because different paths may have a different number of control points, the genotype is of *variable length*. Each point is represented by two numbers representing values across the two axes in the road coordinate system. The bounds for each gene must be known for the optimization process. Every second gene, representing the X-axis component of the control points, can vary from a minimum of 0 to $\sum_{t=k}^{j-1} \left\| P_i^{t+1} - P_i^t \right\|$. Every other gene, representing the Y-axis component of the control points can vary from 0 to 1, in which 0 points to the right boundary and 1 points to the left boundary. This ensures that the GA produces points inside the road. The mapping between the genotype and the phenotype is given by Fig. 6.10.

It is assumed here that all the points are *sorted* as per the values corresponding to the X-axis. This is based on the fact that a vehicle always goes ahead in the direction of the road. Although it may steer by small or large amounts, it never steers large enough to face backwards on the road. Hence, in a valid curve, a latter point always lies ahead in the road from an earlier point. Further, it is assumed that no point lies *behind* the current position of the vehicle X_i, as per the road coordinate system. This is because the vehicle may have crossed the points behind and these are not needed in the curve because the vehicle would not turn back to touch these points.

The road coordinate axis system used for the generation of individuals enables sampling points that lie within road boundaries, unlike the Cartesian coordinate axis system in which the road may be a portion of the entire map, and hence only some of the

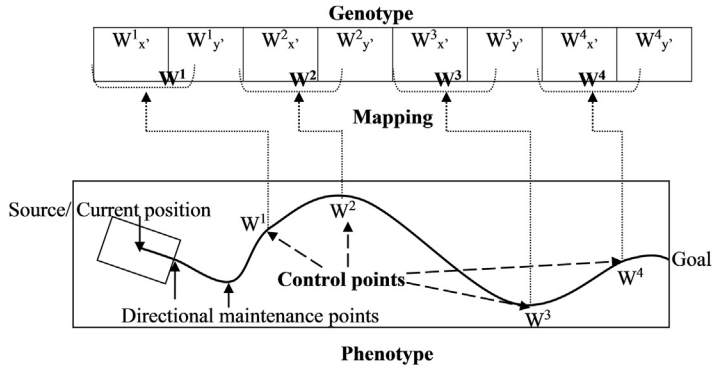

Figure 6.10 Individual representation strategy. Directional maintenance points generate a curve in the heading direction of the vehicle. Bézier control points (variable in number) constitute the genetic genotype, to be optimized by the GA.

points sampled are within road boundaries. In other words, the probability of generation of an obstacle-free point is much higher in the road coordinate axis system. Further, as a result of sorting, any path (in the road coordinate axis system) consists of small path segments between any two points which increase the probability of it being feasible; as compared to a randomly produced path in the Cartesian coordinate axis system wherein two adjacent points would be fairly far apart. By sorting, a number of genotypic representations correspond to a single phenotypic path. This makes the work of the GA easier. A higher probability of generation of a feasible path means the algorithm quickly comes up with a feasible plan which may be optimized in later iterations.

6.4.2 Genetic Operators

Evolution in the GA is performed by a variety of genetic operators. The first operator used is *repair*. This operator ensures that the points are sorted as per the values of the *X*-axis in road coordinate system, all points lie ahead of the current position of the vehicle, and that the individual being considered contains the smallest number of control points. For sorting, the genotype is converted into an array of points consisting of *X* and *Y* coordinates. The array is sorted as per the *X* component. All points in the array behind the current position are deleted, which signify the points that the vehicle has crossed and are no longer needed in the curve representation. The control points are then deleted one by one, until no further deletion results in a better path in terms of its fitness value. This is an implementation of the *shorten* operator. The deletion of points from the main path of the vehicle may well yield a better and shorter path (Xiao et al., 1997). The operation may be time-consuming for most newly generated or random individuals; however, as the algorithm proceeds, with minor or no change in the scenario, the optimized individuals are already short and hence need not result in much consumed time. The operator is shown in Fig. 6.11A.

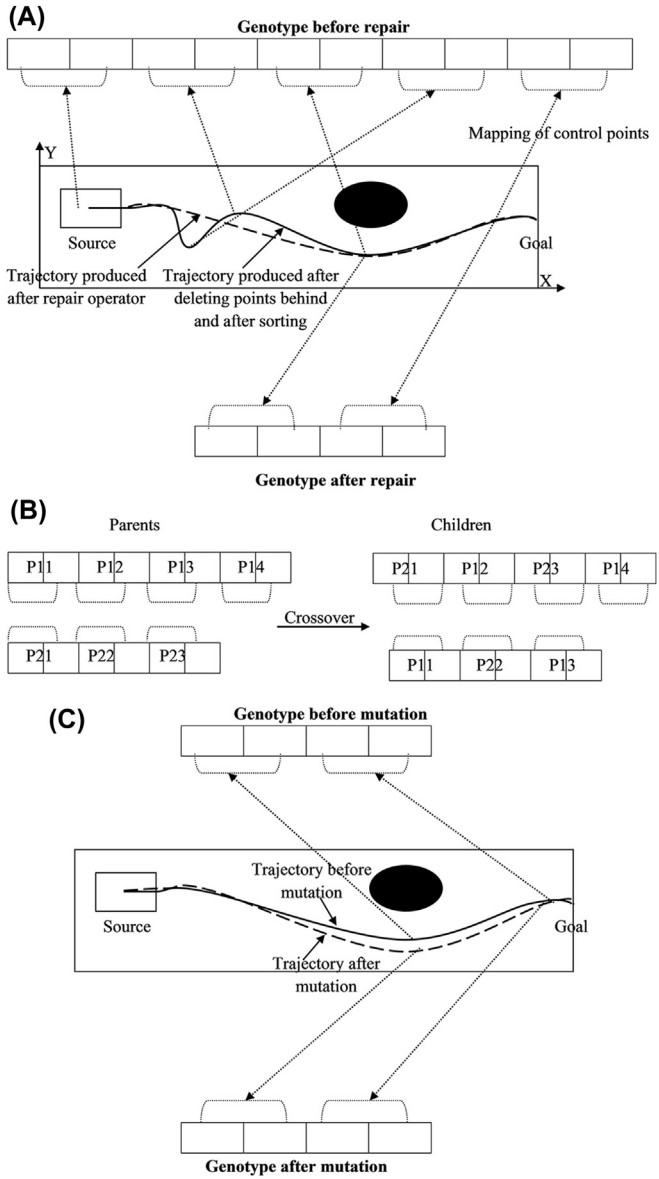

Figure 6.11 Genetic operators. (A) Repair operator. All points are sorted as per the values of the *X*-axis, points behind the vehicle are deleted and control points are deleted till fitness improves. (B) Crossover operator. Because parents are of different length, children are made of mean lengths with random genes exchanged. Symbols denote which gene from parents goes to which child. (C) Mutation operator. Points are moved by a maximum magnitude of 'mutation rate' picked from a Gaussian distribution.

Every deletion of a control point from the path makes the resultant path shorter and smoother. The initial set of points may contain unnecessary twists and turns. However, after this smoothing operation it is ensured that every turn results in the vehicle avoiding an obstacle or remaining within a road boundary. Hence, given a good location of control points, the resultant paths are locally shortest and smoothest. It is assumed that the map does not contain a range of closely packed obstacles such that a generated smoothened path might not be navigable physically by the vehicle. In cases in which many objects occur, the vehicle may initially fail to generate a feasible path, reducing its speed as a result until it is possible to do so.

The second operator used is *insert*, in which random individuals of the highest possible length are generated. These individuals obtain a desirable shape after the application of the repair operator. It is possible that due to scenario changes a trajectory which once looked poor may now be desirable. A converged GA population may not have enough exploratory potential to search for such new and interesting solutions. Hence, random individuals are added to the population pool. The next operator used is the standard *crossover* which generates children by exchanging the characteristics of the parents. Because the number of points (or genes) is variable, the first task is computation of the number of points in the two children (Kala et al., 2010b). The two children have $ceil((n_1 + n_2)/2)$ and $floor((n_1 + n_2)/2)$ points, in which n_1 and n_2 are the number of points in the two parents, *ceil* rounds up to the nearest integer and *floor* rounds down to the nearest integer. These points are randomly taken from the parents, as per the scattered crossover technique. For each vacant point in the first child, a random decision is made whether to take the point from the first parent or the second parent. The corresponding point from the other parent is given to the second child, if vacant positions are available. Leftover points may be used to fill in vacant positions in the children. The operator is shown in Fig. 6.11B.

The next operator used is *mutation*, which changes the value of some point per the mutation rate. In other words, the operator moves the point by some small magnitude over the road to affect the trajectory. The total percentage change that the operator may make is determined by the mutation rate, which is taken from a Gaussian distribution. The relevant operator is shown in Fig. 6.11C. The final operator used is *elite*, which transfers the best individuals of one generation to the next generation. This operator ensures that the best solution found so far is not killed by the other operators, but is retained till the end and can be used for moving the vehicle. Stochastic universal sampling is used to select the individuals for the different operators. Rank-based scaling is used in which the different individuals are assigned weights proportional to their rank in the population pool. The initial population is generated randomly.

Ideally, the optimal individual of a generation is still the optimal individual even after motion of the vehicle as per the travel plan, unless there is a change in P_i^k or P_l^j. This is because the Bézier curve remains the same with the start point shifted to the current position of the vehicle, as the vehicle moves. If the vehicle has crossed a control point, it would be naturally deleted by the repair operator. In case the vehicle physically crosses P_i^k or P_l^j (in other words, when it physically crosses a crossing or there is a change of goal), there is a change of the coordinate system across the movement. Hence, the complete population is reinitialized.

6.4.3 Path Fitness Evaluation

The last part to be considered in the implementation of the GA is the fitness function. Each vehicle cannot be allowed to approach too closely to an obstacle (which may be another vehicle). This allows scope for measurement error as well as enabling a vehicle to make an emergency stop, if necessary, to avoid a collision. Hence, each vehicle must always keep a *clear minimum distance* of d_1 from its front and d_2 from its side. Here, the minimum front distance d_1 and side distance d_2 are given by Eqs [6.5] and [6.6]

$$d_1 = \mathrm{c}_1 v_i^2 \qquad [6.5]$$

$$d_2 = \mathrm{c}_2 v_i^2 \qquad [6.6]$$

in which c_1 and c_2 are constants.

The fitness function is taken to be the *length* of the path. However, at the same time a penalty is added for infeasible paths. The penalty is proportional to the *length of the segments of the infeasible paths* which lie inside obstacles. This creates evolutionary pressures for the GA to minimize the length of the infeasible path up to the point when the infeasible path has been completely eliminated (if possible). The intent is always, of course, to avoid the obstacles, whilst maintaining the *minimum safety distance*. The resultant overall fitness function may therefore be given by Eq. [6.7].

$$\mathrm{Fit} = l + \alpha l_1 + \beta l_2 \qquad [6.7]$$

Here, l is the total length of the Bézier curve, l_1 is the length of an infeasible segment, and l_2 is the length of the path segment in which the minimum safety distance is violated. $\alpha > \beta > 1$. The GA first tries to reduce the length of the infeasible path. If and once a feasible trajectory is obtained, the GA optimizes to reduce the length of the trajectory without the needed safety distance. Then the focus is to minimize the path length.

It is evident that the fitness function of the GA needs to optimize a number of factors considering the vehicle dynamics such as journey time, vehicle speed, path smoothness, distance from obstacles, sharpest turn etc. However, unlike planning for mobile robots, roads are generally smooth enough which in itself practically ensures that any general path smoothing strategy leads to smooth curves. On top of this, vehicles are restricted to low speed as per the traffic laws in force, which means that the allowable speed considering nonholonomic constraints is much higher than the actual speed of travel of the vehicle. For the same reasons there is, in reality, no difference between minimizing time, minimizing path length or increasing smoothness. The safety distance feasibility measure is similar to the normally encountered factor ensuring a minimum distance from obstacles in mobile robotics.

The factors of safety distance (and feasibility) and path length are clearly contradictory. Shorter paths are possible by violating the safety criterion and keeping as many points in the curve as possible very close to the boundary. Unlike other research in this area, this factor is specifically added rather than checking for nonholonomic

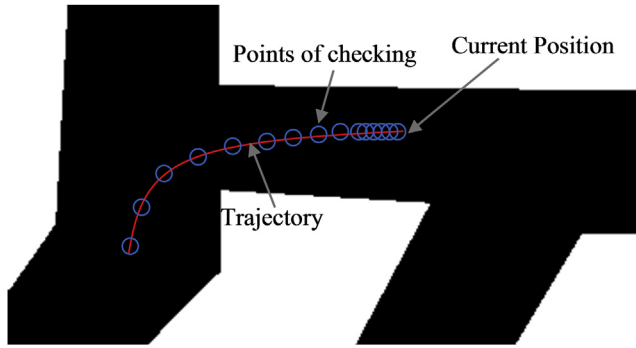

Figure 6.12 Checking of feasibility and distance computation by sampling points. From the entire trajectory generated during planning, only some points are used for feasibility checking and distance computation. These points are closely spaced near the source (current point of the vehicle) and more distance apart later.

constraints or smoothness, and allowing the vehicle to go close to the boundary or obstacles. This is done to account for sensor and actuation uncertainties.

Computation of the feasibility and path length, by point-to-point iteration of the Bézier curve, may be a very costly procedure, and the results of the GA might not be realized in real time. Hence, the computation is performed at points *interleaved by some minimum separation*, in which the separation decreases along the curve. This means that in the initial segment of the curve, each point is checked and, further still, lesser points are checked. The concept is shown in Fig. 6.12.

6.5 Coordination

In decentralized planning, every vehicle is planned separately. Two vehicles, planned separately, may thus have conflicting trajectories if they do not consider each other in their planning. Both vehicles cannot consider each other's trajectories in their planning, because the trajectory computation of the first vehicle is dependent on the availability of the trajectory of the other vehicle and vice versa. The problem is more complicated with the presence of multiple vehicles, wherein any set of vehicles may have conflicting trajectories, whereas the alternative trajectories may be further conflicting. *Coordination* deals with the manner in which such conflicts are handled, or the manner in which vehicles avoid each other; however, this only occurs while each vehicle is planned separately in a decentralized manner.

In addition to the static obstacles, the vehicle needs to ensure that it does not collide with any of the other vehicles. This is done by a space−time-graph approach. Each vehicle maintains a copy of the current plan being followed in the form of a hash map of vehicle positions, hashed with time. Every vehicle considers only the vehicles that are *ahead* of it at the current time, although they can be moving in any direction. Hence, this is an implementation of a priority-based system, in which the priorities are

founded on the position of the vehicles. This is (once again) inspired from natural driving, in which a human driver mainly considers the front view while making their trajectory plan. The purpose of the algorithm is to use the *traffic heuristic* (of overtaking and vehicle following as is presented in Sections 6.5.2 and 6.5.3)-inspired coordination strategy rather than a coordination strategy which jointly optimizes the trajectories of all the vehicles. A traffic inspired heuristic is capable of quickly generating feasible and near-optimal paths.

Coordination dictates the manner in which any two vehicles avoid each other. As per the designed heuristic strategy, a rear vehicle is responsible for avoiding a collision with the vehicle in front. This strategy is, however, noncooperative, and there is no incentive for a front vehicle to aid a rear vehicle in getting a better path. Hence, an additional clause is added that a rear vehicle may request a front vehicle to veer to a particular side, if and by whatsoever amount possible. Whether the request is to be made and, if yes, in what direction depends upon the *overtaking* and *vehicle-following heuristic* discussed in Sections 6.5.2 and 6.5.3. The magnitude by which the vehicle in front veers upon being requested denotes the amount of cooperation which is an algorithm parameter. Further, a vehicle to the rear may request a vehicle in front to alter its speed by some amount, which again depends upon the overtaking and vehicle-following heuristic.

It is expected that after a short time, paths of vehicles will change due to optimization, emergence of other vehicles, obstacles etc. Hence, there is no point in making too far-sighted plans. In natural human driving the focus is on avoiding collisions with nearby vehicles and not avoiding vehicles which are a long way off. Therefore, in this system, a vehicle need not consider collisions with vehicles which are recorded after a large span of time. If a vehicle, faced by a possible collision, is already inside the region where the unsafe distance threshold is breached, slowing of the vehicle will take place.

The algorithm employed here uses a form of communication in which one vehicle can communicate with another vehicle to update its planned path. Initially, when a new vehicle comes into the scenario, it is far from any of the other vehicles and hence even if communication takes a reasonable time to be established, it is acceptable. Once communication is established however, each vehicle knows each other's plans and makes necessary corrections so that an overall feasible travel plan is generated. Plans then only need to be recommunicated when the path actually travelled by a vehicle largely deviates from the one originally communicated at the time of generation of the plan. Hence, the algorithm can handle communication errors to some extent.

The basics of coordination are summarized in Box 6.4. Box 6.5 specifically mentions the key concepts associated with the overtaking and vehicle-following traffic heuristics.

6.5.1 Determining Speed

For a single vehicle operating in a static environment, its trajectory can be computed and traced at any desirable speed without affecting the feasibility. However, operational speed is important to consider for the case of multiple vehicles. The relative

Box 6.4 Key Aspects of Coordination Strategy

- Priority-based coordination
- Only vehicles *ahead* considered
- Cyclic path and speed optimization
  - *Path optimization* by GA
  - *Speed optimization*, increase speed by δ if path feasible, decrease speed by δ if path infeasible
- Cooperation added by *traffic heuristics*
  - Overtake
  - Vehicle following
- Vehicle can *request* another vehicle to cooperate by
  - Slowing down
  - Turning right/left
  - Decision directed by traffic heuristics

Box 6.5 Key Aspects of Traffic Heuristics

- Assess situation for overtaking or vehicle-following behaviours
- If overtaking
  - Give initial turns to the other vehicles to best overtake
  - Alter speeds of the other vehicles to best overtake
  - Cancel overtake if it seems dangerous
- If following a vehicle
  - Give initial turns to the other vehicles to best overtake in the future
  - Alter speeds of the other vehicles to best overtake in the future
  - Initiate overtake if it seems possible

speed of two vehicles determines the coordination or the manner in which the vehicles avoid each other. An optimal coordination plan in the case of multiple robots deals with the optimal planning of the trajectory as well as the optimal planning of speed. The proposed method uses a *path−velocity decomposition* (Kant and Zucker, 1986) scheme wherein path components are optimized by the trajectory planner which works with the assumption that the vehicle would continue to travel with its current constant speed. Considering the needs of the online nature of the algorithm, the velocity component is optimized by a simple mechanism which attempts to assess the scenario and assign the best current speed of the vehicle as the vehicle moves. The attempt is to assign the maximum speed possible to the vehicle, so long as the increased speed does not result in a collision or loss of safety distance.

A simple heuristic rule is used to set the instantaneous speed of the vehicle. At any time the speed of the vehicle may *either be increased by a magnitude of δ or decreased by a magnitude of δ* (Sewall et al., 2011). The speed is always subject to a maximum of

$v_i^{\max}$ which is a property of the vehicle. In the case when the planning routine indicates that the path planned appears to be without any collision and safe distances can be maintained, an attempt is made to increase the vehicle's velocity. However, if the planned trajectory suggests that a collision is likely or that minimum safe distances cannot be achieved, the velocity is decreased. The resultant velocity is hence given by Eq. [6.8].

$$v_i(t+\Delta t) = \begin{cases} \min\left(v_i(t)+\delta, v_i^{\max}\right) & \text{if no penalty} \\ \max(v_i(t)-\delta, 0) & \text{if penalty} \end{cases} \quad [6.8]$$

Considering that the sampled time Δt and the speed interval δ are small, and keeping the path being traced as fixed, it should be possible for a vehicle to quickly obtain the highest possible speed to trace the path. On reaching the highest speed, any attempt to further increase the speed would be turned down as it would make a penalty-free path penalty prone. At the same time, because the current path is without penalty, no decrease in speed would take place. Hence, for a constant path, the vehicle can obtain and maintain its highest speed possible. In reality, as the vehicle speed changes, the algorithm would also modify the path which may better suit the altered speed. For unit iteration, there is a small change in speed which causes a small change in path. Hence, as the vehicle moves, altering speed and path adjustments would be applied leading to convergence which gives a near-optimal plan for navigation.

Having a collision-free near-optimal navigation plan should normally mean that the vehicles stick to it and navigate it using their control mechanisms. However, uncertainties need to be handled. The GA handles uncertainties by maintaining enough separation for the vehicle, at all times, from obstacles, other vehicles and road boundaries. However, actuation errors or uncertainties may make the vehicle lie at a position somewhat different from that expected. Such errors can grow with time. These sensing errors may, however, be corrected with time, as the vehicle approaches obstacles or other vehicles. Hence, the path and speed are constantly adapted to cope with these uncertainties, to make the overall path near optimal and collision free. The experimented results provided are in fact obtained using a lazy control mechanism which is intentionally used to produce large errors. An important reason for using the GA was in fact to overcome these errors. The decision to overtake a vehicle or to follow it instead can be altered based on these uncertainties, which is done outside the GA by a separate decision-making module.

6.5.2 Overtaking

A slower vehicle lying ahead of a faster vehicle, occupying some lateral coverage of the road in use by the faster vehicle behind, is too restrictive for the faster vehicle, forcing it to drive at a lower speed to avoid an accident. *Overtaking* deals with first making the two vehicles lie on different lateral coverage of the road such that the slower vehicle does not block the faster vehicle. It would then be expected that the faster vehicle soon lies completely ahead of the slower vehicle, post which the two vehicles may occupy more favourable lateral positions on the road.

One of the major issues associated with the algorithm is to enable faster vehicles to overtake slower vehicles. As per natural human driving, whenever feasible, a faster vehicle may well overtake a slower vehicle. This makes the travel plan of the faster vehicle more efficient. This section (and Section 6.5.3) gives details on the *heuristics* which lead to the coordination between vehicles. In the algorithm these heuristics are in the form of a vehicle requesting another vehicle to turn, or in the form of a vehicle requesting the other vehicle to slow down. The heuristics are completely derived on the basis of generally observable driving in the absence of speed lanes for comfortable to risky scenarios. The heuristic is based on the philosophy that vehicles need to cooperate, as far as possible, to allow any possible overtake. A key notion here is that not only are the plans monitored and constantly adapted to enable the completion of a successful overtake, the *decision to overtake* itself is monitored and can be altered on sensing a potential threat and reinitiated later.

Initially, the faster vehicle is situated behind the slower vehicle when the overtake procedure starts. The overtake procedure is different for the case when no other vehicle (apart from the vehicle overtaking and the vehicle being overtaken) is present, as opposed to the case when there is an additional vehicle in the overtaking zone.

The first case involves an attempt to overtake by a faster vehicle R_1 which is initially behind a slower vehicle R_2 $\left(v_2^{\max} < v_1^{\max}\right)$. The first task associated with the overtaking procedure is to move the vehicle R_2 leftwards (assuming a driving on the left with overtaking on the right traffic rule) if necessary, in the case when it is in front of R_1. This is important as R_2 does not consider R_1 in its trajectory planning, as R_1 initially lies at the rear of R_2. This move is issued as a *request broadcast* from R_1 to R_2. As an analogy, the process is similar to blowing a horn, indicating an attempt to overtake. The vehicle R_2 considers the request and attempts to move to the left only in the case that the generated path for it to do so is feasible, taking into account possible collisions and minimum safety distances. The opposite rightwards motion of the vehicle R_1 is as per the computation of its trajectory. As long as R_2 moves leftwards, there is more scope for R_1 to overtake, and its trajectory keeps improving with time. The leftwards indicative movements are appended as extra control points in the direction maintenance points in the Bézier-based planning. Eventually, R_1 and R_2 are reasonably far apart when abreast of each other. In a very short time R_1 is able to evolve a trajectory, considering the trajectory of R_2, such that no collision occurs as well as the overtaking procedure taking place.

Ideally, the two vehicles travel as per their own planned trajectories, and the overtaking procedure is completed successfully. However, in this course of time, there may be uncertainties in vehicle control, making the planned trajectories prone to collision. Whenever R_2 detects a possible collision or a minimum safety distance is not maintained, it slows down. This gives greater scope for R_1 to proceed with the overtaking. However, when R_1 detects the absence of a minimum safe distance, it may not slow down, but instead it asks R_2 to slow down. This ensures that R_1 does not lag behind R_2 in the overtaking process. It may again be noted that slowing of R_2 reduces the safe distance which needs to be maintained, which may again contribute towards making overtaking feasible. Many times, whilst overtaking is being carried out, it may be possible that, due to large uncertainties, which may be attributed to a

poor control mechanism, overtaking is completely infeasible. Hence, while overtaking, if R_1 detects that there is a collision (excluding the safe distances) for a few consecutive time steps, it abandons the overtaking procedure. In such a case it proceeds to slow down.

The overtaking mechanism also appears to work well in the scenario of multiple vehicles. However, the procedure may involve the cooperation of other vehicles as well, ie, those not directly involved but which would be affected by the overtaking in terms of disruption to their own trajectory. Here, the first task is to decide whether the vehicle should overtake or not. Suppose that the faster vehicle behind is R_1 and needs to overtake the slower vehicle R_2, with the vehicle R_3 ahead and approaching from the opposite direction. Here, R_3 may in fact be one or many vehicles of the same kind.

Overtaking, at any time of the journey, is regarded as possible if R_1 can draw a feasible trajectory without colliding with R_2 or R_3 as well as maintaining minimum safe distances from them (assuming that R_2 and R_3 already had collision-free trajectories). In such a case, if R_2 or R_3 lie in front of R_1, it would request both of them to move to their left-hand side, respectively, to allow the overtaking procedure of R_2 to occur. Each of these would, however, move as requested only if their generated paths are feasible. The motion of R_1 would then be guided by the planned trajectory of the Bézier curve. As the two vehicles order themselves at their respective left sides, the path of R_1 starts to get both shorter and easier.

As was the case for two vehicles though, it may not ultimately be possible for all three vehicles to navigate as per their respective individual plans and complete the overtaking procedure without a collision. In such a case, a number of actions are required if the overtaking procedure is to go ahead safely. Firstly, vehicles R_2 and R_3 need to slow down when the possible collision is detected. This gives ample time for R_1 to comfortably overtake. Vehicle R_1, on seeing a safety distance not being maintained, may ask R_2 or R_3 to slow down further, dependent on which vehicle it is likely to collide with. However, vehicle R_1 may itself also need to slow down if a collision is detected. If R_2 or R_3 do not cooperate and take plans contrary to the ones desired to safely complete an overtaking procedure, R_1 would soon come into a collision-prone situation and would need to abandon the overtake. The complete overtaking mechanism is shown in Fig. 6.13.

From an analytical perspective, overtaking starts after the vehicles have found a unanimous travel plan by which no collision occurs with wide-enough separations during overtaking. The larger the separation, so the more likely it is that overtaking can be actually carried out. Overtaking can be cancelled at any time if the vehicle has a separation which is not large enough for it to fit in. Once cancelled, the reinitiation of an overtaking procedure may take place at any time when a feasible plan has been constructed. If a vehicle is moved such that sufficiently wide separation is available at all times, overtaking may go ahead with the same speed profiles of vehicles. However, when a vehicle moves, if its separation is not wide enough for it, an attempt is made to increase the separation to the desired threshold by modifying the speed profiles of vehicles. Slowing other vehicles down is usually the best alternative.

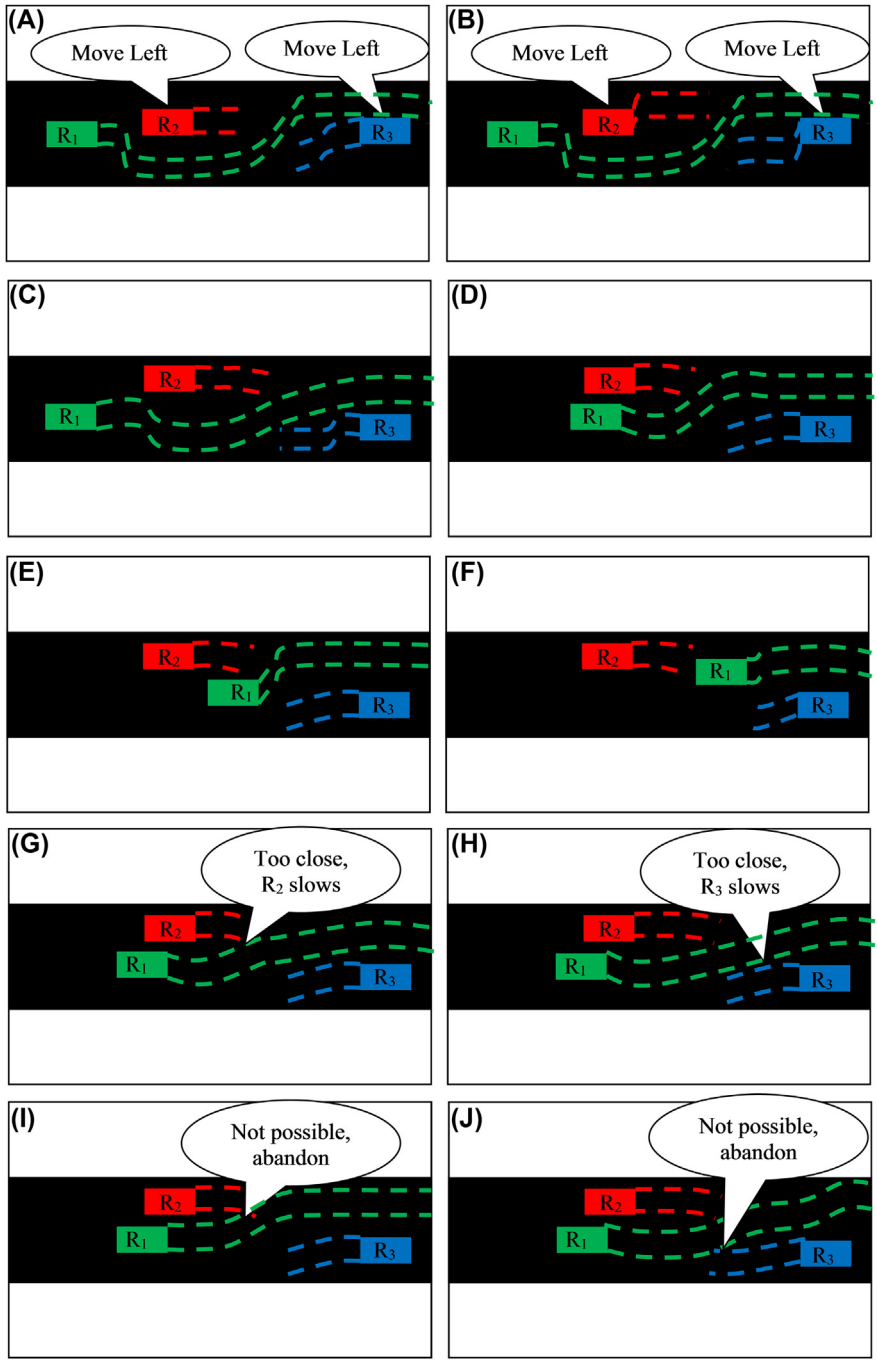

Figure 6.13 Overtaking with three vehicles. (A) The overtaking mechanism starts when R_1 foresees the possibility of overtaking. This is when it is able to construct a feasible and safe trajectory of its motion such that no collision occurs, whilst considering the feasible trajectories of other vehicles. (B) Because R_2 and R_3 are ahead of R_1, these are asked to move in their respective leftwards directions, implemented by giving a leftwards turn to their Bézier curves. (C) R_1 makes a smooth and short trajectory to overtake which is followed by (D) an overtaking manoeuvre. In (E), R_1 has completely overtaken R_2 and in (F) it avoids R_3. (G) to (J) show potential problems which can stop the overtaking procedure from occurring.

6.5.3 Vehicle Following

On many occasions, even though a vehicle to the rear of another may be both capable of and desire a higher speed than the vehicle in front of it, overtaking may not be possible. This can be due to a vehicle in the opposite direction approaching an overtaking point which must be avoided during that specific overtaking action. Conversely, in some scenarios the road may not be broad enough to accommodate vehicle overtaking. In such a case the vehicle simply needs to *slow down and follow* the vehicle ahead. *Vehicle following* is a common traffic behaviour wherein one vehicle constantly adapts its speed to always maintain a safe separation from the vehicle in front, whereas the vehicle may prefer to steer to be roughly behind the vehicle in front. As a result, the two vehicles seem to be following at roughly the same speed with roughly the same lateral positions as long as the vehicle-following behaviour is displayed.

Because the vehicle cannot generate any feasible trajectory at its current speed, it is forced to slow down (ie, if it does not, it will collide with the vehicle in front of it). It keeps slowing in this manner till it generates a feasible trajectory. This feasible trajectory may mostly be available when the vehicle is slow enough not to collide with the vehicle in front. This would happen when the vehicle to the rear is travelling at a speed which is equal to or even slower than the vehicle in front. The vehicle may accelerate at times, thereby increasing its speed; however, when this makes the distance of separation lower than the safe distance, the vehicle would again need to decelerate. Hence, the average speed of the two vehicles would be approximately the same, whilst the scenario remains. If road conditions change, then the situation may change as well.

In such a case, even whilst it is still following the vehicle in front, the vehicle to the rear must be as ready as is possible for overtaking. This means a number of things: it must attempt to push the vehicle being followed as far to its left as possible, any oncoming vehicle as far to its left as possible, to position itself somewhere in the middle, and to start to accelerate ready for the a priori known change in road scenario. This would enable the vehicle to immediately overtake, as soon as it is possible. This mode of operation is summarized in Fig. 6.14.

6.6 Results

The entire planning algorithm was simulated in a custom MATLAB design. The complete scenario was read as an image file drawn using a paint utility tool. Scenario specifications required the number of vehicles, their entry times, entry locations, maximum and initial speeds, acceleration limits etc. All these were read from the scenario specification module. The testing methodology was to individually test each of the features displayed in the algorithm. Accordingly, scenarios were set up which were challenging and exploited the capability of the module under test. First, the algorithm was tested under a number of scenarios, and then analysis of the vehicle behaviour and the algorithm parameters was carried out, which conveys a greater insight into the algorithm's capabilities.

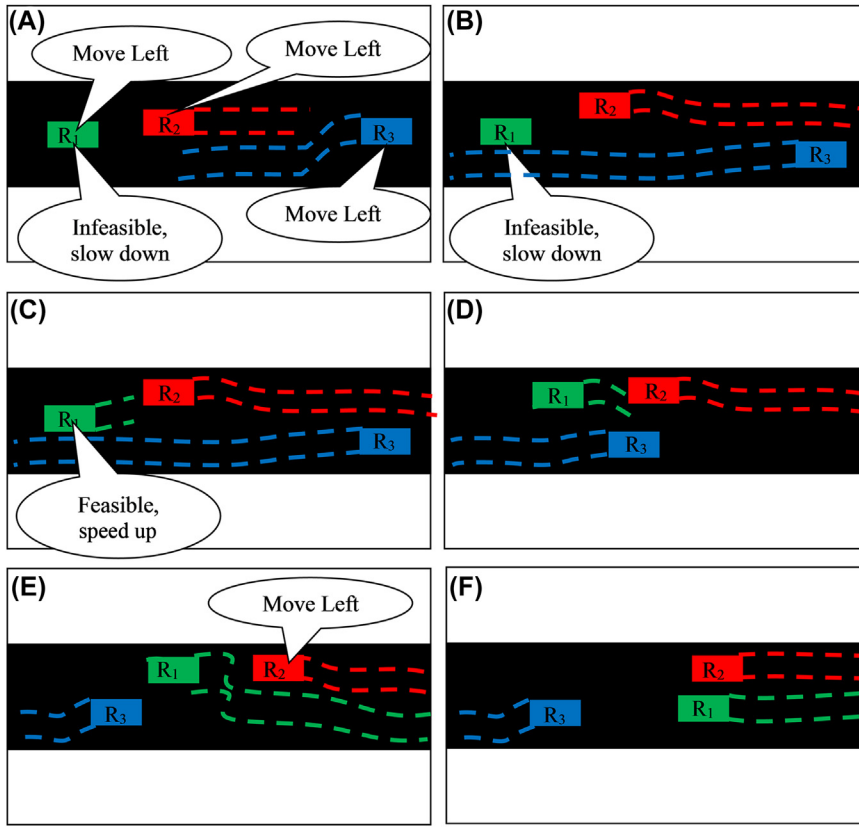

Figure 6.14 Vehicle-following mode of operation. In case vehicle R_1 is unable to compute any feasible trajectory to overtake, it must enter into vehicle-following mode. (A) R_2 and R_3 are requested to move leftwards as they are directly ahead. (B) Vehicle R_1 needs to slow until it is able to compute a feasible trajectory, which would normally mean until it is slow enough to follow R_2. (C) Generation of a feasible trajectory results in an attempt to accelerate. R_1 goes through cycles represented in (B) and (C). R_3 is soon about to pass. (D) R_1 and R_2 move as per the planned trajectories. Once R_3 has passed, R_1 gets ready to overtake R_2 as discussed in Fig. 6.13. (E) First, R_2 is asked to move as far left as possible, whereas R_1 tries to generate feasible and short trajectories. (F) Overtaking is eventually completed.

6.6.1 Higher-level Planning

The first experiment focussed on the higher-level planning of the algorithm. Although finding a route with a given road map is quite a trivial task, the experiment focussed on the ability of the vehicle to drive on the roads, which largely tested the integration between the higher-level planning and lower-level planning. To make it challenging for the lower-level planning to find safe trajectories as per the higher-level path, some obstacles were added on the road. The corresponding path traced by the vehicle is shown

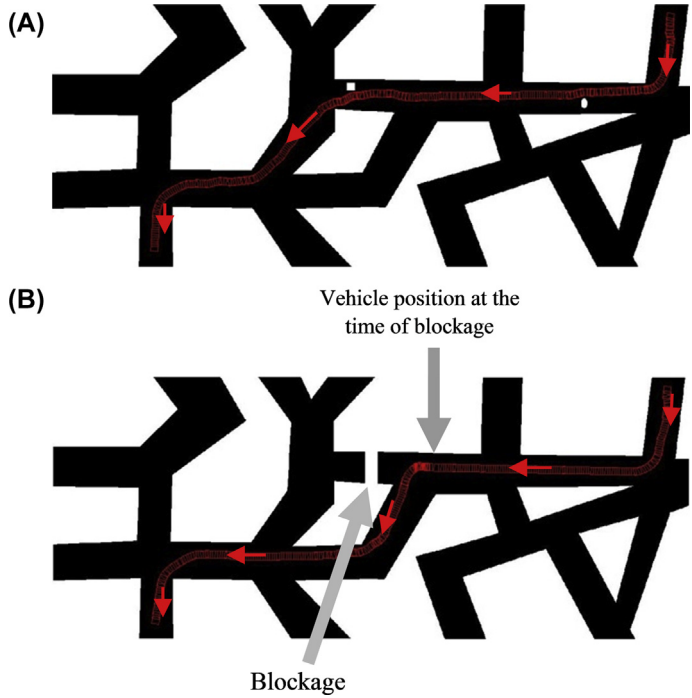

Figure 6.15 Simulation results over a variety of scenarios using GA. A vehicle is made to move in a variety of maps. (A) Tests the ability of the vehicle to steer smoothly around turns and take optimal turns in lesser time, greater speed and smaller path length. (B) Tests the ability of the vehicle to detect blockage and replan the path.

in Fig. 6.15A. This result clearly shows that the vehicle was able to decide which strategy would be the best to avoid an obstacle. The presence of an obstacle though slowed the vehicle's speed a little, which is understandable.

The next feature of the algorithm considered was blockage detection which was tested via another similar experiment. The experiment tested the ability of the lower-level planning algorithm to detect the blockage, the higher-level planning algorithm to replan the path and the ability to make a smooth transition. The vehicle was successfully able to detect the blockage and replan accordingly in real time. A typical traced trajectory is shown in Fig. 6.15B. In this case, an alternative path was available which was used for navigation. It may be seen that there is a sharp change in the path of the vehicle after the blockage is confirmed by the algorithm. Although the algorithm generates infeasible paths, the vehicle's speed is constantly lowered which again helps in making the sharp turns required when a blockage is discovered.

6.6.2 Overtaking

One of the major features of the algorithm is overtaking, which was tested by the next set of experiments. First, some simple experiments are presented to highlight the

various concepts used in the algorithm. Because overtaking is not done at road crossings, a straight-line road was employed as an example indicator. This may be interpreted as a segment of the road in one of the previous scenarios. The first experiment was carried out with two vehicles approaching each other on either side of the road. It was seen that the two vehicles easily coordinated themselves on their left sides. The corresponding trajectory is shown in Fig. 6.16A. Initially, heuristics played a role, in which each vehicle attempted to shift the other vehicle more to its left − this is indicative of the ego type of behaviour exhibited by both vehicles. However, the vehicles soon generated feasible trajectories, avoiding each other with a safe distance between.

The second case is one in which a slower vehicle was initially ahead of a faster vehicle. The vehicle to the rear decides to overtake. This case is shown in Fig. 6.16B−D. The trajectories follow the norms set in the design and it can be seen that the two vehicles are always separated well apart while the overtaking procedure occurs. The scenario was then reversed with the faster vehicle being ahead of the slower vehicle. In such a case an overtaking action will not take place. This was indeed observed by experimentation. The trajectories of the vehicles are shown in Fig. 6.16E. Because the vehicle to the rear had a slower speed, computation of its path was not affected by the vehicle ahead. Hence, it followed more or less exactly the same path as the vehicle in front of it.

Having only two vehicles, to some extent, eases the overtaking process. That said, the main ability of the algorithm to be tested is to be able to carry out a close overtaking

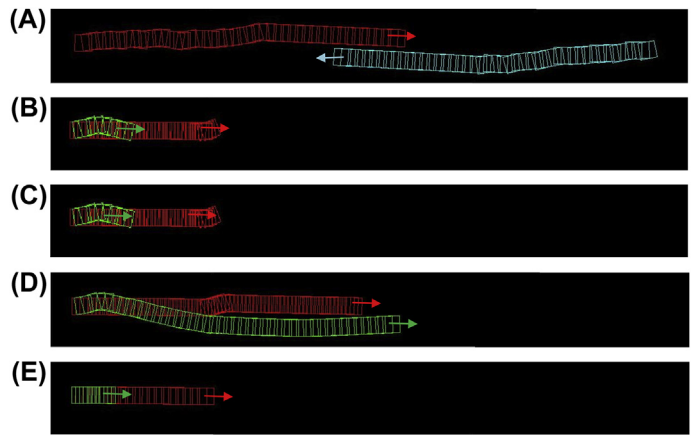

Figure 6.16 Simulation results for multiple vehicle scenarios using GA. Multiple vehicles are moved along a straight road which (A) tests the ability of two vehicles to avoid collision by moving to the left side of the road. The vehicle is further tested for its overtaking ability. In (B), a higher-speed vehicle finds itself behind a slower vehicle. It moves rightwards whereas the vehicle ahead moves leftwards. In (C), the two vehicles are fairly well apart and start moving along their computed paths. In (D), overtaking is regarded as complete, as the vehicle is fast and well ahead. In (E), a slower vehicle is behind a faster vehicle, in which both vehicles follow the same path with no collision possible.

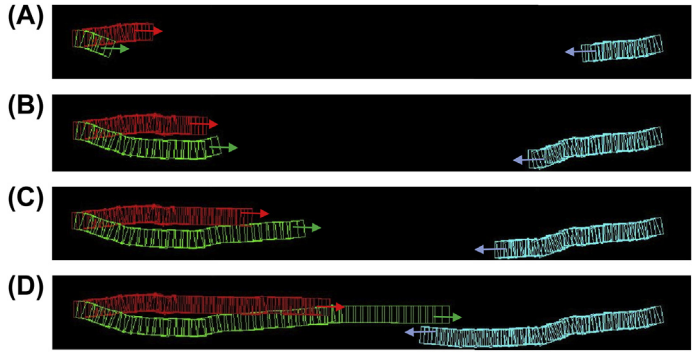

Figure 6.17 Simulation results for overtaking between three vehicles. A faster vehicle needs to overtake a slower vehicle while another vehicle is heading towards the overtaking zone. Overtaking is computed as feasible by the overtaking vehicle. In (A), the two vehicles attempt to move leftwards to give enough room for the overtaking vehicle, which moves rightwards. (B) The overtaking vehicle executes an overtaking trajectory. (C) Overtake of vehicle initially ahead is completed, and (D) finally, the oncoming vehicle is avoided.

procedure. This is investigated here by extending the situation to three vehicles, with the third vehicle approaching from the opposite direction. The oncoming vehicle therefore restricts overtaking, which must be completed before any potential collision with the oncoming vehicle. The speed of the oncoming vehicle is kept low enough to make overtaking feasible, but high enough to make overtaking close, which the algorithm must perform. The vehicles have errors associated with their movements, which must be overcome in the small overtaking gap available. The trajectories of this motion are shown in Fig. 6.17A–D. It can be seen that the feasibility was correctly computed, based on the time of overtake initiation. The initial overtaking trajectory appears similar to the case with only two vehicles; however, the challenge was to timely align the overtaking vehicle to avoid a collision with the oncoming vehicle. In this case, a sufficient margin of safety was available after the overtaking procedure.

6.6.3 Vehicle Following

Vehicle following is a complex behaviour which not only tests the ability of the algorithm to enable one vehicle to follow another vehicle, but also to be prepared for any potential overtaking procedure in the future. Balancing the two acts is a challenge which the algorithm must deal with appropriately. This is tested by an experiment similar to overtaking, with the difference that the speed of the vehicle is adjusted to make an overtaking action infeasible. The trajectories are shown in Fig. 6.18A–D. It may be seen that in this case the vehicle first followed the slower vehicle, primarily requiring adjustment (slowing) of speed and location of drive. The vehicle in this case also attempted to be as far to the right as possible. However, in case it came close to the oncoming vehicle, it also attempted to drift to the left – balancing the need to overtake

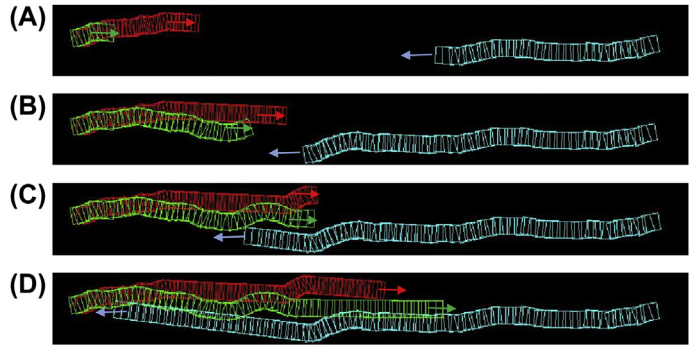

Figure 6.18 Simulation results for vehicle-following behaviour. A faster vehicle behind a slower vehicle is unable to generate a feasible overtaking trajectory due to an oncoming vehicle and hence needs to slow down and follow the vehicle in front. (A) The vehicle attempts to align itself in the middle of two vehicles, giving enough room for the oncoming vehicle to pass by. (B) The vehicle is expected to drift leftwards whereas the oncoming vehicle does the same to avoid collision. (C) The oncoming vehicle is completely avoided and overtaking may now take place, which is initiated. The vehicle moves rightwards to execute an overtaking trajectory. The other vehicle cooperates by moving leftwards. (D) Overtaking is completed.

with that of avoiding the oncoming vehicle. As soon as the oncoming vehicle had passed, the vehicle in question then proceeded to overtake the slower vehicle. It can be seen that there was an early (failed) attempt to overtake, but at all times the specified minimum safe distances had to be maintained.

6.6.4 Vehicle Behaviour Analysis

Here, the behaviour of the vehicle is metrically assessed, when placed in a variety of situations on the road. The intent is to monitor how the vehicle manages its speed and the available road width for its navigation. The factors of efficiency and safety can be contradictory, and yet these need to be balanced for optimal travel.

Efficiently making turns was one of the prime inspirations behind the use of Bézier curves. The vehicle performance is assessed in this respect. Measurements were taken of the desired trajectory for the vehicle around a curve (in the form of a map), the time of completion of the map and the distance travelled at a variety of maximum speeds. The corresponding plots are given in Fig. 6.19. At lower speeds, the vehicle travelled near the inner edge of the curve and hence the distance traversed was low; however, this distance increased as the speed increased and the vehicle started travelling towards the outer edge of the curve. Subsequent increases of speed had no effect on distance travelled. This may sound counterintuitive at first; however, it must be remembered that at very high speeds the paths generated to safely travel around a curve are infeasible and the vehicle needs to lower its speed until such a path becomes feasible. All units are arbitrary and specific to the simulation tool. These relate to the real-world units by constants.

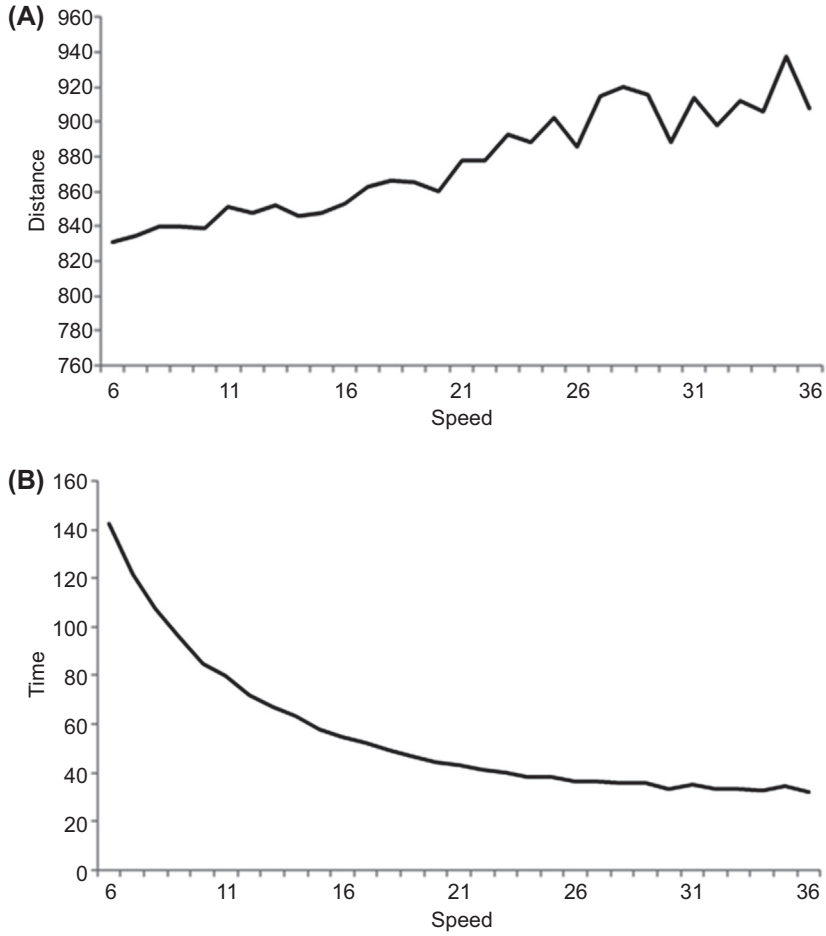

Figure 6.19 Relation between (A) distance of travel with speed (B) time of travel with speed.

In terms of time taken to travel round a curve, results are interesting. Increased speeds usually imply less travel time; however, the increased distance travelled round a curve implies a longer travel time. However, speed is dominant; the time of travel decreases with increase in speed, although this decrease is not uniform. At higher speeds, the vehicle first slows down which reduces the average speed (differentiating between average and maximum speed) to some degree.

6.6.5 Genetic Algorithm Parameter Analysis

The major parameters associated with the algorithm are within the GA which significantly affects overall algorithm performance. Because the solution needs to be returned in real time, it is not possible for the GA to have either a large number of generations for convergence or much time for its optimal solution search. For this reason,

some GA operators were designed to constantly add diversity to the population pool, whilst others attempted to tune the trajectory and converge. Because both these aspects become important in different situations, the analysis cannot be done on the basis of a set of benchmark situations, in which clearly one or other strategy would be of more use. Considering the operation of the repair operator and its ability to quickly produce reasonable paths and small probabilities of invalidation of paths, it can be argued that less effort is needed to add diversity and more effort to tune the trajectory. The same strategy is used in the algorithm.

In such a context, the population size of the GA becomes a major parameter of the algorithm, which needs to be high enough to assure a feasible solution in a low number of generations and low enough to assure a small minimum execution time. In all experiments, this value was fixed at 20 because of the following: At many times the population is reinitialized for the algorithm, and hence it is necessary that the solution generated in a single generation of the algorithm (which makes the immediate move of vehicle) is feasible and near optimal (if not optimal). Hence, first, one specific curved scenario is considered and the number of individuals required to generate a decent feasible path are analysed. The total number of individuals or paths generated was increased and the best path length was noted. The corresponding graph is given in Fig. 6.20. It required at least six individuals for the generation of feasible paths, avoiding collisions and ensuring minimum safe distances. As the number of individuals was increased, a small decrease in path length was visible. The improvement obtained for a much higher number of individuals was minimal. Experimental evaluations reveal that use of a Cartesian coordinate axis system for the GA individual representation gives infeasible solutions for the part of the graph shown in Fig. 6.20. This gives an indication of the superiority of the road coordinate axis system over the Cartesian coordinate axis system.

Similarly, another experiment was performed with a different number of obstacles on the curved path. The average number of individuals required for the generation of a feasible path was noted. The graph so generated is given in Fig. 6.21. An increase in the number of obstacles means a greater difficulty in the generation of the feasible path

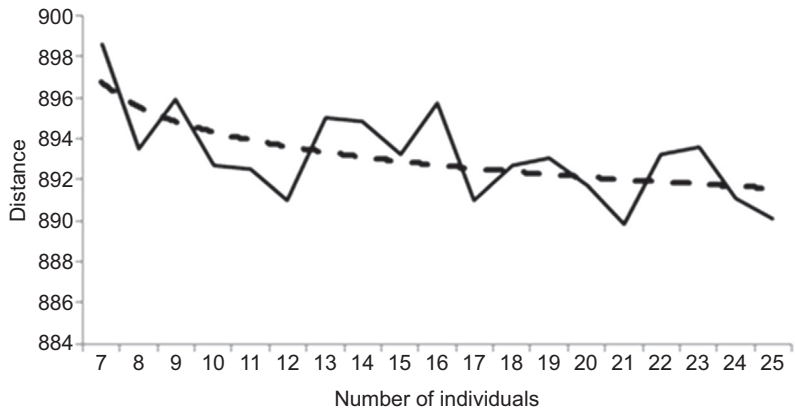

Figure 6.20 Relation between distance of travel and the number of random individuals used.

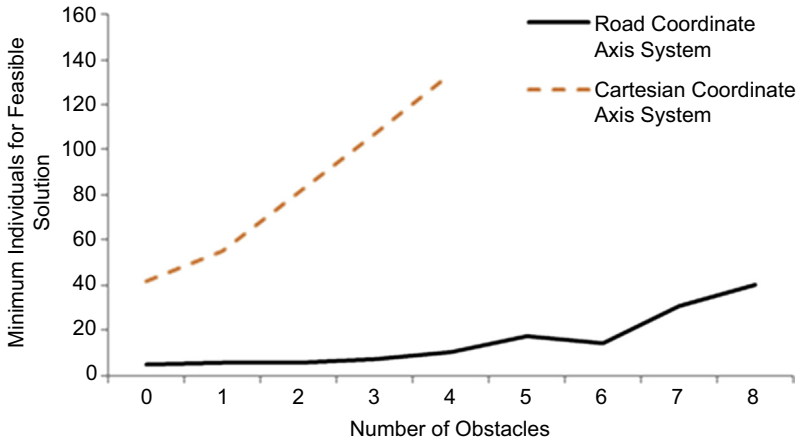

Figure 6.21 Relationship between the number of individuals needed for the generation of a feasible path and the number of obstacles in the path.

as the vehicle needs to make more turns to avoid obstacles. Further, it means fewer possibilities of feasible paths. The increase in difficulty also depends on the place where the obstacle is placed. An increase in obstacles hence increased the number of individuals required, with this increase being more at higher obstacle paths. Any further fitting of obstacles on the path was not possible due to limited road size.

Interestingly, at one instance the number of individuals required actually decreased with an increase in the number of obstacles. This appears to be due to the fact that the effect of one extra obstacle was to guide the Bézier curve in a manner that another obstacle was (automatically) avoided. From both of these experiments it can be inferred that a population size of 20 may be a good choice for the simulation as it stands. This ensures the generation of feasible paths for reasonably complex scenarios, whilst the paths are also short enough for simpler scenarios.

In Fig. 6.21, the road coordinate axis system is compared to a Cartesian coordinate axis system, in which the points used for individual representation of the GA were initially generated randomly in the road segment being planned. In this experiment, the aim was to test to what degree the road coordinate axis system was better (or worse) than the Cartesian system. Results clearly indicate a significant improvement in the number of individuals required when using the proposed coordinate axis system. Further, for more than four obstacles the Cartesian coordinate system was unable to generate a feasible solution even for a very high number of individuals. Hence, the Cartesian coordinate system is limited to planning with a reasonably small number of obstacles only.

6.7 Summary

Planning and coordination of multiple autonomous vehicles on a road network is an exciting problem. From the point of view of a single vehicle, the challenge lies in

effective strategic planning at the top level to obstacle avoidance at the lower level. The real-time nature of the problem, however, eliminates the possibility of developing fancy solutions which are time costly. Unlike most mobile robotics studies, the cost of collisions in vehicular systems is likely to be high and must be avoided by all means. The presence of multiple vehicles stresses the need for coordination strategies between vehicles at all times such that no collision occurs between the vehicles and such that each vehicle has a fair path along which to move forwards at its own desired speed as much as possible. The practical high diversity in speeds of the vehicles, together with a relatively small workplace make the problem very different from existing (central processing) solutions in the research domain of mobile robotics.

In this chapter, GA was used to solve the problem of navigation of vehicles within the road segment under the assumption of communication. The greatest challenge was generating quality solutions in near-real time. The solution takes some inspiration from traffic rules, which theoretical analysis and experimental observations verify to be optimal, considering a large number of scenarios and the presence of uncertain incidents. These traffic heuristics, used with an evolutionary algorithm for the generation of Bézier curves, are able to navigate the vehicle in a variety of scenarios. The experimented scenarios range from straight roads and roads with a large number of obstacles to close overtaking and infeasible overtaking. This more or less covers the variety of scenarios that the vehicle may face in real life. Uncertainties are the key element that makes software simulations differ from real world experiments. Here, uncertainties were introduced by making the vehicle move as per its own kinematic model, and not necessarily the planned trajectory. The planner at times did generate sharp curves, which could not be followed by the vehicle. These errors were corrected with time.

References

Bartels, R.H., Beatty, J.C., Barsky, B.A., 1998. Bézier Curves. An Introduction to Splines for Use in Computer Graphics and Geometric Modelling. Morgan Kaufmann, San Francisco, CA, pp. 211–245.

Bennewitz, M., Burgard, W., Thrun, S., 2001. Optimizing schedules for prioritized path planning of multi-robot systems. In: Proceedings of the 2001 IEEE International Conference on Robotics and Automation, Seoul, Korea, pp. 271–276.

Bennewitz, M., Burgard, W., Thrun, S., 2002. Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots. Robotics and Autonomous Systems 41 (2–3), 89–99.

Chakraborty, J., Konar, A., Chakraborty, U.K., Jainm, L.C., 2008. Distributed cooperative multi-robot path planning using differential evolution. In: Proceedings of the IEEE World Congress on Evolutionary Computation, pp. 718–725.

Chu, K., Lee, M., Sunwoo, M., 2012. Local path planning for off-road autonomous driving with avoidance of static obstacles. IEEE Transactions on Intelligent Transportation Systems 13 (4), 1599–1616.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., 2001. An Introduction to Algorithms, second ed. MIT Press, Cambridge, MA.

Garcia, M.A.P., Montiel, O., Castillo, O., Sepulveda, R., Melin, P., 2009. Path planning for autonomous mobile robot navigation with ant colony optimization and fuzzy cost function evaluation. Applied Soft Computing 9 (3), 1102−1110.

Kala, R., 2012. Multi-robot path planning using co-evolutionary genetic programming. Expert Systems With Applications 39 (3), 3817−3831.

Kala, R., Shukla, A., Tiwari, R., 2010a. Fusion of probabilistic A* algorithm and fuzzy inference system for robotic path planning. Artificial Intelligence Review 33 (4), 275−306.

Kala, R., Shukla, A., Tiwari, R., 2010b. Dynamic environment robot path planning using hierarchical evolutionary algorithms. Cybernetics and Systems 41 (6), 435−454.

Kala, R., Shukla, A., Tiwari, R., 2011. Robotic path planning using evolutionary momentum based exploration. Journal of Experimental and Theoretical Artificial Intelligence 23 (4), 469−495.

Kala, R., Warwick, K., 2014. Heuristic based evolution for the coordination of autonomous vehicles in the absence of speed lanes. Applied Soft Computing 19, 387−402.

Kant, K., Zucker, S.W., 1986. Toward efficient trajectory planning: the path-velocity decomposition. The International Journal of Robotics Research 5 (3), 72−89.

Kapanoglu, M., Alikalfa, M., Ozkan, M., Yazıcı, A., Parlaktuna, O., 2012. A pattern-based genetic algorithm for multi-robot coverage path planning minimizing completion time. Journal of Intelligent Manufacturing 23 (4), 1035−1045.

Klancar, G., Skrjanc, I., 2010. A case study of the collision-avoidance problem based on Bernstein−Bézier path tracking for multiple robots with known constraints. Journal of Intelligent Robotic Systems 60 (2), 317−337.

Lepetic, M., Klancar, G., Skrjanc, I., Matko, D., Potocnik, B., 2003. Time optimal path planning considering acceleration limits. Robotics and Autonomous Systems 45, 199−210.

Lian, F.L., Murray, R., 2003. Cooperative task planning of multi-robot systems with temporal constraints. In: Proceedings of the 2003 IEEE International Conference on Robotics and Automation, pp. 2504−2509.

Paruchuri, P., Pullalarevu, A.R., Karlapalem, K., 2002. Multi agent simulation of unorganized traffic. In: Proceedings of the ACM 1st International Joint Conference on Autonomous Agents and Multiagent Systems, New York, pp. 176−183.

Potter, M.A., 1997. The Design and Analysis of a Computational Model of Cooperative Coevolution. George Mason University, Fairfax, Virginia (Ph.D. thesis).

Sewall, J., van den Berg, J., Lin, M.C., Manocha, D., 2011. Virtualized traffic: reconstructing traffic flows from discrete spatio-temporal data. IEEE Transactions on Visualization and Computer Graphics 17 (1), 26−37.

Stanley, K.O., Miikkulainen, R., 2004. Competitive coevolution through evolutionary complexification. Journal of Artificial Intelligence Research 21 (1), 63−100.

Szlapczynski, R., Szlapczynska, J., 2012. On evolutionary computing in multi-ship trajectory planning. Applied Intelligence 37 (2), 155−174.

Wang, M., Wu, T., 2005. Cooperative co-evolution based distributed path planning of multiple mobile robots. Journal of Zhejiang University - Science A 6 (7), 697−706.

Xiao, J., Michalewicz, Z., Zhang, L., Trojanowski, K., 1997. Adaptive evolutionary planner/navigator for mobile robots. IEEE Transactions on Evolutionary Computing 1 (1), 18−28.

Xidias, E.K., Azariadis, P.N., 2011. Mission design for a group of autonomous guided vehicles. Robotics and Autonomous Systems 59 (1), 34−43.